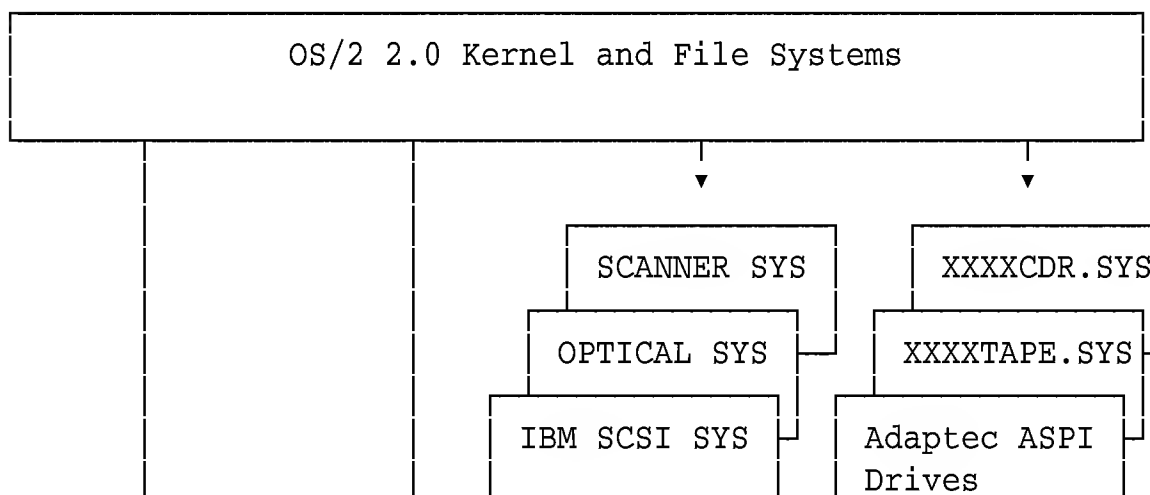

Introduction to DASD, SCSI, and CD-ROM Programming Interfaces

This reference defines the OS/2* 2.1 programming interfaces to support original equipment manufacturer (OEM) direct access storage devices (DASD), small computer system interface (SCSI) devices, and compact disk read only memory (CD-ROM) devices.

The programming interfaces described in this reference provide the following benefits:

- o Device drivers can be written in the C programming language.
- o The development of new DASD, SCSI, and CD-ROM support for unique device interfaces is expedited by reducing the amount of new code required and the complexity of that code.
- o Facilitate development of a new DASD, SCSI, or CD-ROM driver for a specific bus interface.
- o Relatively complex OS/2 kernel interfaces are replaced with a single interface.
- o Independent development organizations are better able to reuse existing DASD device driver code.
- o OS/2 2.1 is better equipped for installing, starting, and operating on a broad range of Intel** 80386SX-compatible workstations.

The following figure illustrates the organization of the new code:



OS2DASD.DMD	OS2CDROM.DMD	OS2SCSI.DMD	OS2ASPI.DMD
DASD Mgr	CDROM Mgr	SCSI DD Mgr	ASPI DD Mgr

TOSHCDS1.FLT

HITCDS1.FLT

CDROM Filters

IBM2SCSI.ADD

IBM2ADSK.ADD

IBM2FLPY.ADD

IBM1S506.ADD

IBM1FLPY.ADD

OEMXXXX.ADD

IBM Drivers

Additional OEM
Drivers

The following types of device drivers are included in this reference:

- o Device managers
- o Adapter device drivers
- o Filter device drivers

A *device manager (DM)* is a hardware-independent module that services the standard OS/2 request packet interface. An *adapter device driver* is a hardware-dependent module and is a member of the lowest layer in the device-driver hierarchy. The adapter device driver to device manager interface is designed such that an adapter device driver is little more than a *state machine* responsible for moving

blocks of I/O between system memory and a target device.

A *filter device driver* differs from an adapter device driver in that it normally does not manage hardware directly. See Filter Device Drivers and Using Filter Device Drivers for details about filter device drivers.

Device Managers

Device managers provide a uniform interface between their clients and adapter device drivers. Device manager clients normally are an OS/2 installable file system or the OS/2 kernel but can be other device drivers.

The interface between a device manager and the adapter device drivers managed is defined in this reference. The interface between device managers and the clients they service is defined by the client's interface specification.

IBM provides the devices managers shown in the following table:

Device Manager	Client	Client Specification
OS2DASD.DMD	OS/2 File Systems	OS/2 Physical Device Driver Reference
OS2SCSI.DMD	SCSI.SYS option drivers	OS/2 SCSI Device Driver Specification
OS2ASPI.DMD	ASPI option drivers	Advanced SCSI Programming Interface
OS2CDROM.DMD	CD-ROM File System	OS/2 CD-ROM Interface

Adapter Device Drivers

Adapter device drivers provide a uniform software interface to the hardware devices they manage. A device driver's external interface is defined in this reference.

Adapter device drivers for the following industry-standard interfaces are included in the OS/2 2.1 product:

Device Driver	Supported Devices
IBM1S506.ADD	ISA ST-506 and AT-compatible IDE drives
IBM1FLPY.ADD	ISA removable media drives
IBM2ADSK.ADD	ABIOS fixed drives
IBM2SCSI.ADD	ABIOS SCSI adapters
IBM2FLPY.ADD	ABIOS removable media drives
IBMINT13.I13	INT 13H BIOS DASD devices

Additional adapter device drivers for other OEM interfaces might be included in the OS/2 operating system.

Filter Device Drivers

Filter device drivers are a special class of device drivers that provide the following:

- o Generic value-added services, such as data stripping or encryption
- o Device-specific services, such as adjusting and altering the command stream between a device manager and an adapter device driver to support a particular type of device

The interfaces between device managers and filter device drivers are identical to the interfaces between device managers and ordinary adapter device drivers. Filter drivers differ from ordinary drivers in that they normally do not manage hardware directly; instead, they monitor the stream of commands between a device manager and regular adapter device drivers.

Filter device drivers to support the following devices are included in the OS/2 2.1 product:

Filter	Supported Devices
HITCDS1.FLT	SCSI-1 Hitachi CD-ROM drives
TOSHCD1.FLT	SCSI-1 Toshiba CD-ROM drives
NECCDS1.FLT	SCSI-1 NEC CD-ROM drives
SONYCDS1.FLT	SCSI-1 Sony CD-ROM drives

Installation of OS/2, DASD, SCSI, and CD-ROM Device Drivers

The key design points of this OEM DASD, SCSI, and CD-ROM device support package include:

- o The ability to install and, subsequently, start up from a DASD device that requires an OEM-specific adapter device driver interface
- o An installation process that is transparent to the end-user (that is, it requires no interaction on the part of the end-user)

This chapter describes the strategy developed to address these design points and the responsibilities of a device driver supplier in order to participate in this strategy.

Using the BASEDEV Keyword

A *base device driver* performs I/O during the OS/2 kernel initial load sequence. There are a number of operational differences between base device drivers and installable device drivers. See Adapter Device Driver Development Considerations for a description of how this affects adapter device driver development.

The *BASEDEV* keyword new with the OS/2 2.0 operating system, loads a base device driver into the operating system. Its syntax is as follows:

```
BASEDEV=      filename
              arguments
```

Unlike the *DEVICE=* statement, the *BASEDEV=* statement must not contain either drive or path information. The root directory of the startup partition is searched first for the specified file name, followed by the \OS2 directory of the startup partition. (In the startup sequence, the OS/2 operating system cannot process drive or path information at the point where *BASEDEV=* statements are processed. If drive or path information is included there, an error is generated.)

Also, unlike the *DEVICE=* statement, the file-name extension of the file being loaded has a special meaning. *BASEDEV=* statements are not necessarily processed in the order in which they appear in your CONFIG.SYS file. The extension of each *BASEDEV=* file name is examined; then, *BASEDEV=* statements are processed in the order indicated by the following figure.

```
BASEDEV= Load Ordering by File Extension
.SYS (processed first)
.BID
.VSD
.TSD
.ADD
.I13
.FLT
.DMD (processed last)
```

Files with other file-name extensions are not loaded.

If several *BASEDEV=* statements load file names with the same extension, those files are loaded in the order in which they are encountered in the CONFIG.SYS file.

OS/2 System Installation

When the OS/2 operating system is first loaded from the installation diskettes, the following adapter device drivers and device managers are loaded:

Adapter Device Driver	Supported Device Managers
OS2DASD.DMD	OS/2 DASD manager
OS2CDROM.DMD	OS/2 CD-ROM manager
IBM1FLPY.ADD	ISA removable media driver
IBM1S506.ADD	ISA ST-506 driver
IBM2FLPY.ADD	ABIOS removable media driver
IBM2ADSK.ADD	ABIOS DASD driver
IBM2SCSI.ADD	ABIOS SCB driver
IBMINT13.I13	Generic INT 13h driver

Additional drivers supporting other OEM interfaces also can be present.

When each device driver initializes, it attempts to determine whether its target hardware adapter is present. If the hardware interface is recognized, the driver completes its initialization and, subsequently, is ready to manage I/O operations during OS/2 system installation. If the hardware interface is not recognized, the device driver will fail the initialization with the *Quiet Fail* flags set. *Quiet failure* prevents the generation of failure messages on the workstation display.

Hardware interfaces that are not recognized by any of the drivers on the OS/2 initialization diskette are driven by the generic INT 13h adapter device (IBMINT13.I13) during installation. The IBMINT13 driver determines whether the previously loaded adapter device drivers have claimed at least as many fixed disks as indicated by the BIOS fixed disk count (0:475). The IBMINT13 driver will attempt to manage the remaining fixed disks. Consequently, to install and initially load the OS/2 operating system from an OEM adapter, it is important for the OEM to ensure that the IBMINT13 adapter device driver works properly with the

OEM's adapter BIOS.

The OS/2 operating system can be installed and loaded on drives with BIOS IDs hex 80 or higher, provided that the OEM BIOS supplies INT 13h support for these drives.

OEM Adapter Device Driver Installation

OEM adapter device drivers are installed within the framework of the OS/2 DDINSTAL utility. The driver developer is responsible for supplying two modules (in addition to the adapter device driver) used by DDINSTAL - an adapter *presence-check function* and a *device driver profile*. DDINSTAL uses these modules to automatically detect the presence of OEM hardware interfaces and to install the corresponding drivers without user intervention.

Presence-Check Function

A *presence-check function* is a Ring 3 (nonprivileged) EXE program that determines whether a given hardware interface is present on a workstation. The module returns **0** when the specific interface is detected and **1** when the interface is not detected. For these modules to identify installed OEM adapters, Ring 0 services are provided by the device driver TESTCFG.SYS. TESTCFG provides the following IOCTL services for OEM adapter presence-check modules:

- o Determines CPU host bus type
- o Reads adapter ROM space
- o Executes *IN/OUT* instruction
- o Reads Micro Channel adapter POS IDs
- o Reads EISA adapter IDs

See Adapter Presence-Check Services (TESTCFG.SYS) for details on the TESTCFG device driver services.

Note: Be sure to write adapter presence-check modules to avoid disruption or conflicts with other installed host adapters.

Device Driver Profiles

A *device driver profile* is a file with a DDP extension containing a script that is interpreted by the OS/2 DDINSTALL utility. The device driver profile defines which files to copy from the installation diskettes to the target directories and specifies how the CONFIG.SYS file will be updated.

Refer to the *Physical Device Driver Reference* for specification of the DDINSTALL utility and the device driver profile language.

The DDINSTALL utility has been extended to support execution of the presence-check function and to conditionally process the DDP file, based on the return code from the presence check. To enable this support, the DDINSTALL utility now interprets the PRESENCECHECK keyword.

To use this new DDINSTALL feature, create a DDP file for the installation of your adapter driver, using the existing TITLE, CONFIG, and FILES keywords. Then, add a line to the DDP of the form:

```
:PRESENCECHECK  
<filename>
```

where <filename> is the name of the presence-check function.

When the DDP is interpreted by DDINSTALL, that utility first scans the DDP for the PRESENCECHECK keyword. If the keyword is found, the corresponding EXE module is executed. Then, the entire DDP file is either processed or ignored, based on the outcome of the presence-check function.

A device driver profile for a hypothetical OEM-323x SCSI adapter could look like the following example. The file name would be OEM323x.DDP and the contents would be as follows:

```
*****  
*                                                                 *  
* This is a device driver profile for a SCSI adapter.           *  
* DDINSTALL would use this profile to automatically install the *  
*****
```

```
* target device support. The complete profile is processed only *
* when the OEM323x.EXE program returns 0, indicating that the *
* OEM-323x adapter is actually installed in the workstation. *
* *
*****
```

```
:PRESENCECHECK      * Check for the presence of an OEM-323x.
OEM323x.EXE          * This might query POS IDs using TESTCFG.
```

```
*****
* The remainder of this file is processed only if *
* OEM323x.EXE indicates detection of the OEM-323x adapter.*
*****
```

```
:TITLE
Device driver profile for the OemTec OEM-323x OS/2 2.0 Adapter Device
Driver
```

```
:CONFIG      * Add this line to CONFIG.SYS
BASEDEV=OEM323x.ADD
```

```
:FILES
OEM323x.ADD \OS2\OEM323x.ADD
      * Move this file from the installation
      * diskette to the \OS2 directory on the
      * target partition.
```

Processing Presence-Check Functions and DDP Files

OEM adapter device drivers that are packaged in the OS/2 product are installed near the end of the OS/2 system installation. At this point in the installation process, the DDP files for each OEM adapter device driver are evaluated by the DDINSTAL interpreter. This processing is completely automatic and transparent to the end user.

Use the same DDINSTAL framework for adapter device drivers that you distribute directly. Include the driver file, presence-check function, and DDP file on a reference diskette for the OEM adapter. The end user can install the adapter device support from the reference diskette, after the OS/2 operating system is loaded, by selecting *Device Driver Install* from the OS/2 System Setup folder. The installation of the device support will proceed automatically.

Adapter Device Driver Development Considerations

Adapter device drivers are packaged as 16-bit OS/2 device drivers. This chapter describes how adapter device drivers differ from installable OS/2 device drivers.

Loading and Initialization

Adapter device drivers are loaded using the BASEDEV= statement in CONFIG.SYS. The processing of these statements occurs before the operating system is fully initialized. The adapter device driver writer must be aware of the following differences between installable device drivers and adapter device drivers:

- **Adapter device drivers initialize at Ring 0 rather than Ring 3.**

Generally, this does not cause any problems. However, adapter device drivers cannot use the DOS~~xxx~~ APIs available to installable device drivers during initialization. To display a message, an adapter device driver must use the DevHlp_Save_Message service.

- **INIT request packet command code**

The INIT request packet command code for all base device drivers (which include all adapter device drivers) is hex 1B rather than hex 0.

- **Device Driver Header**

An adapter device driver must identify itself as a participant in the adapter device driver strategy by setting the following bits to **1** in the device driver header. The bit-numbering convention is that bit 15 is the most significant bit in a WORD, and bit 31 is the most significant bit in a DWORD.

- Device attribute field - Bits 15, 8, 7

Bit 15 indicates CHARACTER device driver. Bits 8 and 7 define *driver* as a Level 3 device driver, which indicates usage of the *DWORD capabilities bit strip* in the header file.

- Capabilities Bit Strip - Bit 3

Bit 3 indicates that the driver is participating in the adapter device driver strategy which, in turn, selects an alternate INIT request packet format from the kernel.

- **INIT request packet format**

The INIT request packet for a driver that has identified itself as an adapter device driver (through bits set in the device driver header as just described) corresponds to the RPINITIN structure defined in REQPKT.H, supplied with the *IBM Device Driver Source Kit For OS/2*. The *InitArgs* member of the RPINITIN structure points to the following structure, defined in DSKINIT.H in the kit.

```

typedef struct _DDD_PARM_List {          /* DDPL                                *,
    USHORT      reserved1;                /* Reserved                                *,
    USHORT      disk_config_table;        /* Address of config table                *,
    USHORT      reserved2;                /* Reserved                                *,
    USHORT      cmd_line_args;            /* Address of command line parm          *,
    USHORT      machine_config_table;     /* Address of config info                 *,
} DDD_Parm_List, FAR *PDD_Parm_List;

```

By following the appropriate pointers in the DDD_Parm_List, the driver writer can obtain BASEDEV= command-line parameters, as well as information collected during system initialization.

- **Adapter device drivers process a limited set of OS/2 kernel request packets.**

With the exception of the OS/2 system kernel initialization request packet just described and vendor-defined IOCTLs, adapter device drivers must reject all other kernel request packets. The primary interface to adapter device drivers is defined in this reference.

- **Adapter device drivers register their entry points using the DevHlp service.**

Adapter device drivers register their main entry points with the OS/2 kernel using the DevHlp_RegisterDeviceClass service. See DASD, SCSI, and CD-ROM Device Manager Interface Specification for details. The table of registered entry points is available to other adapter device drivers and device managers that can call an adapter device driver directly.

- **Adapter device drivers must declare a valid character device name in their headers.**

The OS/2 kernel treats the name in the adapter device driver header as a valid character device name. Adapter device drivers must end their device names with a dollar sign (\$) to avoid conflict with valid file names.

- **Adapter device drivers must fail *quietly* when hardware is not found.**

Adapter device drivers should check for the presence of their hardware interface at initialization time. If it is not found, the adapter device driver must set the ERROR_I24_QUIET_INIT_FAIL flags (as defined in BSEERR.H) in the *Status* field of the request packet.

Operation

Adapter device drivers receive commands through an I/O request block (IORB) entry point. The format of IORB commands received by an adapter device driver is defined in this reference.

Adapter device drivers have full use of both the 16-bit and 32-bit DevHlp services defined in OS/2 operating system. Although the adapter device driver to DM interface is 16-bit, adapter device drivers can manipulate 32-bit objects with assembly subroutines.

The service request entry point of an adapter device driver can be called in either kernel (also known as *task*) or interrupt contexts. Consequently, *an adapter device driver must never block while servicing a request after it has completed initialization*. (An adapter device driver *can* block at initialization.)

Service requests that involve time delays normally are initiated by the adapter device driver; then, the adapter device driver immediately returns to its caller. Service request completion is indicated to the caller using asynchronous callback notification.

Command-Line Parameters

To facilitate the parsing of command-line parameters, and to help encourage uniformity in command-line syntax, a *parser/tokenizer* is provided in the *IBM Device Driver Source Kit For OS/2*. In addition, a command-line syntax definition is provided in Adapter Device Driver Command-Line Parameters.

The output of the parser/tokenizer is a stream of tokens that represents the contents of the command line. The parser/tokenizer performs preliminary syntactical checks on the command line and indicates the results of these checks by the return code. However, the adapter device driver must ensure that the tokenized parameters' values are acceptable. The adapter device driver is responsible for displaying error messages as appropriate.

OEMs can modify the parser and included tables to add their own adapter-unique flags and parameters.

DASD, SCSI, and CD-ROM Device Manager Interface Specification

The IBM OS/2 2.1 DASD and SCSI device manager interface consists of direct call commands and Device Helper (DevHlp) services.

Direct Call Command Interface

All direct call commands are issued by the device managers (OS2DASD.DMD and OS2SCSI.DMD) or filter device drivers to an adapter device driver's registered entry point, with a global pointer to the Input/Output Request Block (IORB), as follows:

C Language Syntax

```
#include <iorb.h>

VOID (FAR * ADDEntryPoint) (piorb);

PIORB   piorb;           /* Far pointer to the IORB control block */
```

Assembly Language Syntax

```
#include <iorb.inc>

; ** ES:BX = IORB Pointer
PUSH    es                ; IORB Segment
PUSH    bx                ; IORB Offset
CALL    dword ptr AddEntryPoint ; Call adapter device driver
ADD     sp, 4              ; Clean-up stack
```

Results

The results of the command are returned in the IORB.

The following table categorizes and lists the direct call commands used for the DASD and SCSI device manager interface:

Command Type	Commands
CONFIGURATION	GET_DEVICE_TABLE COMPLETE_INIT
UNIT_CONTROL	ALLOCATE_UNIT DEALLOCATE_UNIT CHANGE_UNITINFO
GEOMETRY	GET_MEDIA_GEOMETRY SET_MEDIA_GEOMETRY GET_DEVICE_GEOMETRY SET_LOGICAL_GEOMETRY
EXECUTE_IO	READ READ_VERIFY READ_PREFETCH WRITE WRITE_VERIFY
FORMAT	FORMAT_MEDIA FORMAT_TRACK FORMAT_PROGRESS
UNIT_STATUS	GET_UNIT_STATUS CHANGELINE_STATE GET_MEDIA_SENSE GET_LOCK_STATUS
DEVICE_CONTROL	ABORT RESET SUSPEND RESUME LOCK_MEDIA UNLOCK_MEDIA EJECT_MEDIA
ADAPTER_PASSTHRU	EXECUTE_SCB EXECUTE_CDB

DevHlp services introduced with the OS/2 2.0 operating system to support this

strategy include:

- o RegisterDeviceClass
- o GetDOSVar

IORB Control Blocks

All direct call command control blocks are defined in the IBM-supplied IORB.H and IORB.INC Include files. (See I/O Request Block - C Definitions.) The following sections, which describe the commands and their associated control blocks, are written from both C and assembler programmers' points of view, with references to the actual Include files and field names.

IORB General Format

The IORB is the main control block for all direct call commands. To accommodate varying command-specific data, there are eight types of IORBs, one per *CommandCode*, as shown in the following table.

IORB Type	CommandCode
IORB_CONFIGURATION	IOCC_CONFIGURATION
IORB_UNIT_CONTROL	IOCC_UNIT_CONTROL
IORB_GEOMETRY	IOCC_GEOMETRY
IORB_EXECUTE_IO	IOCC_EXECUTE_IO
IORB_FORMAT	IOCC_FORMAT
IORB_UNIT_STATUS	IOCC_UNIT_STATUS
IORB_DEVICE_CONTROL	IOCC_DEVICE_CONTROL
IORB_ADAPTER_PASSTHRU	IOCC_ADAPTER_PASSTHRU

Each IORB consists of a common I/O Request Block Header (IORBH data structure), followed by unique command-specific data, as shown in the following table.

Field Name	C Type	Length	Description
Length	USHORT	DW	Length of IORB
UnitHandle	USHORT	DW	Unit handle
CommandCode	USHORT	DW	Command code
CommandModifier	USHORT	DW	Command modifier
RequestControl	USHORT	DW	Flags

Status	USHORT	DW	Status
ErrorCode	USHORT	DW	Error code
Timeout	ULONG	DD	Completion timeout
StatusBlockLen	USHORT	DW	Length of status info
pStatusBlock	NPBYTE	DW	Pointer to status info
Reserved_1	USHORT	DW	Reserved
pNxtIORB	PIORB	DD	Pointer to next IORB
NotifyAddress	(*PFN) ()	DD	Notification address
DMWorkSpace[20]	UCHAR	DB(20)	Reserved
ADDWorkSpace[16]	UCHAR	DB(16)	adapter device driver work area

On entry to the driver:

Length

is set to the total length of the IORB (IORBH plus *Command-Specific Data*) in bytes.

UnitHandle

identifies the adapter device driver's unit for which the request is intended. The adapter device driver must assign a unique *UnitHandle* in the DEVICETABLE UNITINFO structure for each of the units it manages. Refer to the *IOCC_CONFIGURATION CommandCode* section for additional information.

CommandCode/CommandModifier

contains the direct call commands. These commands are grouped by *CommandCode* as shown in the following table. The *CommandCode* field defines the IORB; the *CommandModifier* field selects the actual operation within a specified *CommandCode*. For details on each of the commands, refer to their corresponding *CommandCode* sections.

CommandCode

CommandModifier

IOCC_CONFIGURATION	IOCM_GET_DEVICE_TABLE IOCM_COMPLETE_INIT
IOCC_UNIT_CONTROL	IOCM_ALLOCATE_UNIT IOCM_DEALLOCATE_UNIT IOCM_CHANGE_UNITINFO
IOCC_GEOMETRY	IOCM_GET_MEDIA_GEOMETRY IOCM_SET_MEDIA_GEOMETRY IOCM_GET_DEVICE_GEOMETRY IOCM_SET_LOGICAL_GEOMETRY
IOCC_EXECUTE_IO	IOCM_READ IOCM_READ_VERIFY IOCM_READ_PREFETCH IOCM_WRITE IOCM_WRITE_VERIFY
IOCC_FORMAT	IOCM_FORMAT_MEDIA IOCM_FORMAT_TRACK IOCM_FORMAT_PROGRESS
IOCC_UNIT_STATUS	IOCM_GET_UNIT_STATUS IOCM_GET_CHANGELINE_STATE IOCM_GET_MEDIA_SENSE IOCM_GET_LOCK_SENSE
IOCC_DEVICE_CONTROL	IOCM_ABORT IOCM_RESET IOCM_SUSPEND IOCM_RESUME IOCM_LOCK_MEDIA IOCM_UNLOCK_MEDIA IOCM_EJECT_MEDIA
IOCC_ADAPTER_PASSTHRU	IOCM_EXECUTE_SCB IOCM_EXECUTE_CDB

RequestControl

contains flags which control the processing of the IORB, as shown in the following table.

Flag	Description
IORB_ASYNC_POST	Command-completion protocol. If set, this flag indicates that the <i>NotifyAddress</i> field is valid and that the adapter device driver should call this routine when the IORB request is completed.
IORB_CHAIN	IORB chaining. If set, this flag indicates that the <i>pNxtIORB</i> field is valid and that there is a chained IORB command to service.
IORB_CHS_ADDRESSING	I/O addressing format. If set, this flag indicates that the command's <i>RBA</i> field is in the format defined by the CHS_ADDR structure. This bit should be set only for diskette controllers.
IORB_REQ_STATUSBLOCK	Request for status information. If set, this flag indicates that the <i>StatusBlockLen</i> and <i>pStatusBlock</i> fields are valid and that the adapter device driver should return the command's associated status information.
IORB_DISABLE_RETRY	No error retry. If set, this flag indicates that the adapter device driver should not retry the request if a processing error occurs.

For more information about chained IORBs (IORB_CHAIN), see Adapter Device Driver Interface Questions and Answers.

Status

equals 0 on entry. Upon exit from the adapter device driver, *Status* contains flags to indicate the command's completion status. (See the following table.)

Flag	Description
IORB_DONE	Processing complete. If set, this flag indicates that the adapter device driver has completed processing the request.
IORB_ERROR	Error encountered. If set, this flag indicates that an error occurred while processing the request. This flag should not be set if the error was successfully recovered by the adapter device driver.
IORB_RECOV_ERROR	Recoverable error. If set, this flag indicates that, although an error occurred, the adapter device driver successfully recovered through retries.
IORB_STATUSBLOCK_AVAIL	Status information returned. If set, this flag indicates that the adapter device driver has returned status information in the buffer defined by <i>pStatusBlock</i> .

ErrorCode

equals 0, on entry. On exit from the driver, it contains the command's completion error code. This field is valid only if the IORB_ERROR flag in the *Status* field is set. The error codes are summarized in Error Handling.

Timeout

contains the maximum number of seconds the driver will permit for command completion before timing out. If this field is set to 0, the timeout value assigned is the default set by the driver. If this field is set to -1, the timeout value assigned is infinite. The timeout period is measured from the last valid contact (interrupt) with the target device. Therefore, if the device interrupts periodically within the timeout interval, the interval is reset after each interrupt.

StatusBlockLen

contains the size of the block of storage, in bytes, for the driver to return status information (*pStatusBlock*). This field is valid only if the IORB_REQ_STATUSBLOCK flag is set in the *RequestControl* field.

pStatusBlock

contains a near pointer to a block of storage (length = *StatusBlockLength*), allocated by the caller, for the driver to return status information. On exit from the driver, the storage area contains status information. This field is valid only if the IORB_REQ_STATUSBLOCK flag is set in the *RequestControl* field. The format of information in the status block depends on the class of adapters the driver supports. For SCSI devices, see IORB Status Block for more information.

Note: The pointer to the status block is a 16-bit near pointer. The status block must reside in the same segment as the IORB.

Reserved_1

is reserved for use by the device manager and must not be modified by the adapter device driver.

pNxtIORB

contains a far pointer to the next IORB for chained commands. This field is valid only if the IORB_CHAIN flag is set in the *RequestControl* field.

NotifyAddress

contains a far pointer to the notification routine to be called when the request has completed successfully or aborted due to error conditions. This field is valid only if the IORB_ASYNC_POST flag is set in the *RequestControl* field. The notification routine should be called with a far pointer to the command's IORB.

C Language Syntax

```
(FAR *piorb->NotifyAddress) (piorb);
```

Assembly Language Syntax

IORB CommandCode Format

The IORB CommandCode format is defined in the following section.

IOCC Configuration

The IOCC_CONFIGURATION *CommandCode* consists of all the *CommandModifiers* responsible for returning information about the characteristics of the devices supported by the driver, as follows:

IOCM_COMPLETE_INIT

Indicates that the driver can complete its initialization phase.

In the interval between driver initialization and receipt of this IORB, the device driver must not disable its INT 13h BIOS support because this support is needed to load other components of the operating system.

IOCM_GET_DEVICE_TABLE

Returns the DEVICETABLE structure in the buffer supplied by the caller. DEVICETABLE contains detailed information on each adapter and the associated units supported by the adapter device driver.

Remarks

Support: Mandatory
Called By: OS2DASD.DMD, other device manager, or filter device driver
Context of Call: TASK

Note: Any adapter device driver that registers by way of the RegisterDeviceClass DevHlp must process this IORB and return a valid DEVICETABLE, even if the driver supports 0 adapters.

Format of IORB

- o IORB Type
 - IOCB_CONFIGURATION
- o IORB Fields
 - *CommandCode*
 - o IOCC_CONFIGURATION
 - *CommandModifiers*
 - o IOCM_COMPLETE_INIT
 - o IOCM_GET_DEVICE_TABLE
 - Valid RequestControl Flags

- o IORB_ASYNC_POST

IORB_CONFIGURATION Description

This section defines the IORB_CONFIGURATION control block and the following associated structures:

DEVICETABLE	Table of supported devices
ADAPTERINFO	Adapter characteristics
UNITINFO	Unit characteristics

DEVICETABLE Structure Overview

pDeviceTable DEVICETABLE

pAdapter[0]

pAdapter[1]

. . .

pAdapter[N]

ADAPTERINFO 0

ADAPTER 0

UNIT 0

UNITINFO[0]

UNITINFO[1]

. . .

. . .

UNITINFO[N]

UNIT N

ADAPTERINFO 1

ADAPTER 1

UNIT 0

UNITINFO[0]

UNITINFO[1]

. . .

. . .

UNITINFO[N]

UNIT N

ADAPTERINFO N

ADAPTER N

UNIT 0

UNITINFO[0]

UNITINFO[1]

. . .

. . .

UNITINFO[N]

UNIT N

IORB_CONFIGURATION

Field Name	C Type	Length	Description
iorbh	IORBH	DB(68)	IORB header
pDeviceTable	FAR *PDEVICETABLE	DD	Device table
DeviceTableLen	USHORT	DW	Length of table

On entry to the driver:

iorbh

See IORB General Format.

pDeviceTable

contains a far pointer to a block of storage (length = *DeviceTableLen*), allocated by the caller, for the driver to return the DEVICETABLE.

DeviceTableLen

contains the length of the block of storage, in bytes, for the driver to return the DEVICETABLE (*pDeviceTable*).

DEVICETABLE

Field Name	C Type	Length	Description
ADDLevelMajor	UCHAR	DB	ADD major level
ADDLevelMinor	UCHAR	DB	ADD minor level
ADDHandle	USHORT	DW	ADD index
TotalAdapters	USHORT	DW	Number of adapters
pAdapter[N]	NPADAPTERINFO	DW(N)	AdapterInfo pointers

On exit from the driver:

ADDLevelMajor/ADDLevelMinor

defines the level of support the adapter device driver is written to. A driver written to this specification (IBM OS/2 2.0 Support Level), should set the fields as follows:

Field Name	Value
ADD_Level_Major	ADD_LEVEL_MAJOR
ADD_Level_Minor	ADD_LEVEL_MINOR

ADDHandle

contains the adapter device driver's index returned by the *RegisterDeviceClass* DevHlp.

TotalAdapters

defines the number of adapters the device driver supports.

pAdapter[N]

contains an array of near ADAPTERINFO pointers. The number of elements in the array is determined by the *TotalAdapters* field.

AdapterInfo

Field Name	C Type	Length	Description
AdapterName[17]	UCHAR	DB(17)	ASCIIIZ name
Reserved	UCHAR	DB	Reserved. Must be 0.
AdapterUnits	USHORT	DW	Number of units
AdapterDevBus	USHORT	DW	Device bus types
AdapterIOAccess	UCHAR	DB	Host I/O type
AdapterHostBus	UCHAR	DB	Host bus type
AdapterSCSITargetID	UCHAR	DB	Target ID
AdapterSCSILUN	UCHAR	DB	Logical unit number

AdapterFlags	USHORT	DW	Flags
MaxHWSGList	USHORT	DW	Max HW s/g elements
MaxCDBTransferLength	ULONG	DD	Max CDB data transfer length
UnitInfo[N]	UNITINFO	DD (N)	Unit information

On exit from the driver:

AdapterName[17]

contains the ASCIIZ name string of the adapter. This name is used by the caller for diagnostic purposes.

Reserved

contains a 0. This is a 16-bit alignment byte.

AdapterUnits

contains the number of units supported by this adapter.

AdapterDevBus

defines the adapter-to-device bus protocol used, as shown in the following table.

Protocol	Description
AI_DEVBUS_ST506	DASD - ST506 CAM-I
AI_DEVBUS_ST506_II	DASD - ST506 CAM-II
AI_DEVBUS_ESDI	DASD -ESDI
AI_DEVBUS_FLOPPY	DASD - Diskette
AI_DEVBUS_SCSI_1	SCSI - Version-I
AI_DEVBUS_SCSI_2	SCSI -Version-II
AI_DEVBUS_SCSI_3	SCSI - Version-III
AI_DEVBUS_OTHER	Protocol not listed.

AI_DEVBUS_NONSCSI_CDROM non-SCSI CD-ROM interface

One protocol should be elected. The AdapterDevBus protocol values can be OR'd with one or more modifier bits as listed in the following table.

Modifier	Description
AI_DEVBUS_FAST_SCSI	Fast SCSI bus timings
AI_DEVBUS_8BIT	8-bit bus width
AI_DEVBUS_16BIT	16-bit bus width
AI_DEVBUS_32BIT	32-bit bus width

AdapterIOAccess

defines the adapter-to-host I/O data transfer capabilities, as shown in the following table.

Flag	Description
AI_IOACCESS_BUS_MASTER	1st-party DMA adapter
AI_IOACCESS_PIO	Programmed INs/OUTs
AI_IOACCESS_DMA_SLAVE	2nd-party DMA adapter
AI_IOACCESS_MEMORY_MAP	Memory-mapped I/O
AI_IOACCESS_OTHER	I/O access not listed.

AdapterHostBus

defines the adapter-to-host bus type used, as shown in the following table.

Type	Device Connection
AI_HOSTBUS_ISA	ISA
AI_HOSTBUS_EISA	Extended ISA

AI_HOSTBUS_uCHNL	Micro-channel
AI_HOSTBUS_OTHER	Bus type not listed.
AI_HOSTBUS_UNKNOWN	Bus type unknown.

Width	Description
AI_HOSTBUS_8BIT	8-bit bus
AI_HOSTBUS_16BIT	16-bit bus
AI_HOSTBUS_32BIT	32-bit bus
AI_HOSTBUS_64BIT	64-bit bus
AI_HOSTBUS_UNKNOWN	Bus width unknown.

Note: One bus type should be set with one bus width OR'd in.

AdapterSCSITargetID

contains the target ID for the SCSI adapter. For non-SCSI devices, this field should be set to 0.

AdapterSCSILUN

contains the logical unit number for the SCSI adapter. For non-SCSI devices, this field should be set to 0.

AdapterFlags

IOCC_UNIT_CONTROL

The IOCC_UNIT_CONTROL *CommandCode* consists of all the *CommandModifiers* responsible for controlling the ownership of a unit. The following table describes the *CommandModifiers*.

CommandModifier	Description
IOCM_ALLOCATE_UNIT	Assigns ownership of the specified unit to the caller. A unit must be allocated prior to accepting any other direct call commands. Once allocated, a unit cannot be assigned to another owner until that unit is deallocated. It is the responsibility of the owner to coordinate sharing of a unit.
IOCM_DEALLOCATE_UNIT	Removes the caller's ownership of the specified unit. Once deallocated, a unit can be assigned to another owner.
IOCM_CHANGE_UNITINFO	Modifies the specified unit's UNITINFO portion of the DEVICETABLE structure with the information passed by the caller.

Remarks

Support: Mandatory
Called By: OS2DASD.DMD, other device manager, or filter device driver
Context of Call: TASK

Format of IORB

- o IORB Type
 - IORB_UNIT_CONTROL
- o IORBH Fields
 - *CommandCode*

- o IOCC_CONFIGURATION
- *CommandModifiers*
 - o IOCM_ALLOCATE_UNIT
 - o IOCM_DEALLOCATE_UNIT
 - o IOCM_CHANGE_UNITINFO
- Valid *RequestControl* Flags
 - o IORB_ASYNC_POST

IORB_UNIT_CONTROL Description

This section defines the IORB_UNIT_CONTROL control block. (See the table below.)

Field Name	C Type	Length	Description
iorbh	IORBH	DB(68)	IORB header
Flags	USHORT	DW	Flags
pUnitInfo	PUNITINFO	DD	Pointer to UnitInfo
UnitInfoLen	USHORT	DW	Length of UnitInfo

On entry to the driver:

iorbh

See IORB General Format.

Flags

contains a 0.

pUnitInfo

contains a far pointer to a buffer containing modified unit characteristics, in the format defined by the UNITINFO structure. The adapter device driver uses this information to update the unit's UNITINFO structure in the DEVICETABLE. This field is valid only for the IOCM_CHANGE_UNITINFO *CommandModifier*.

Note: A device driver can access the UNITINFO structure provided by the IOCM_CHANGE_UNITINFO IORB at any time. The caller, therefore, must not invalidate or release the passed UNITINFO structure on successful completion of this IORB request.

UnitInfoLen

contains the length, in bytes, of the UNITINFO buffer (*pUnitInfo*) passed to the driver. This field is valid only for the IOCM_CHANGE_UNITINFO *CommandModifier*.

On exit, the driver sets the *Status* and *ErrorCode* fields of the IORBH to reflect the results of the IOCC_UNIT_CONTROL request.

Return Codes

Following is a list of the IOCC_UNIT_CONTROL error codes:

- IOERR_CMD_SYNTAX
- IOERR_CMD_SW_RESOURCE
- IOERR_UNIT_ALLOCATED
- IOERR_UNIT_NOT_ALLOCATED

For a detailed description of all the return codes, see Error Handling.

IOCC_GEOMETRY

The IOCC_GEOMETRY *CommandCode* consists of all the *CommandModifiers* responsible for setting and returning information about the capacity of a unit.

The *CommandModifiers* are described in the following table:

CommandModifier	Description
IOCM_GET_MEDIA_GEOMETRY	Returns the geometry of the current media in a drive. For non-removable media devices, the geometry returned must be identical to the geometry returned by IOCM_GET_DEVICE_GEOMETRY.
IOCM_SET_MEDIA_GEOMETRY	Informs the adapter device driver of the required media geometry in preparation for formatting. This command is mandatory only for standard diskette media.
IOCM_GET_DEVICE_GEOMETRY	Returns the device geometry compatible with INT 13h BIOS function 08h. If the INT 13h support for a device provides translation, the INT 13h geometry of the device must be returned with the BIOS translation performed within the driver. That is, the driver must emulate any INT 13h translation performed by BIOS.
IOCM_SET_LOGICAL_GEOMETRY	Indicates that the geometry recorded in the file system tables on the media does not match the physical media

geometry reported by the device driver.
The driver should convert RBA to CHS addresses according to the geometry passed in this IORB, rather than using the media geometry the driver is reporting. The device driver should stop performing this translation if a media change indication is detected.
Support of this command is mandatory only for standard diskette media.

Remarks

Support: Mandatory (See *CommandModifiers* for exceptions.)
Called By: OS2DASD.DMD, other device manager, or filter device driver
Context of Call: TASK, INTERRUPT

Format of IORB

- o IORB Type
 - IORB_GEOMETRY
- o IORBH Fields
 - *CommandCode*
 - o IOCC_GEOMETRY
 - *CommandModifiers*
 - o IOCM_GET_MEDIA_GEOMETRY
 - o IOCM_SET_MEDIA_GEOMETRY
 - o IOCM_GET_DEVICE_GEOMETRY
 - o IOCM_SET_LOGICAL_GEOMETRY
 - Valid *RequestControl* Flags
 - o IORB_ASYNC_POST
 - o IORB_REQ_STATUSBLOCK
 - o IORB_DISABLE_RETRY

IOCC_GEOMETRY Description

This section defines the IORB_GEOMETRY and GEOMETRY control blocks. (See the table below.)

Field Name	C Type	Length	Description
iorbh	IORBH	DB(68)	IORB header
pGeometry	PGEOMETRY	DD	Pointer to GEOMETRY
GeometryLen	USHORT	DW	Length of GEOMETRY data

On entry to the driver:

iorbh

See IORB General Format.

pGeometry

contains a far pointer to the block of storage (length = *GeometryLen*) allocated by the caller for the GEOMETRY.

GeometryLen

contains the size of the block of storage, in bytes, for the GEOMETRY structure (*pGeometry*).

GEOMETRY Description

Field Name	C Type	Length	Description
TotalSectors	ULONG	DD	Number of sectors
BytesPerSector	USHORT	DW	Bytes per sector
Reserved	USHORT	DW	Reserved
NumHeads	USHORT	DW	Number of heads
TotalCylinders	ULONG	DD	Number of cylinders
SectorsPerTrack	USHORT	DW	Number of sectors per track

On entry to the driver for SET *CommandModifiers*, and on exit from the driver for GET *CommandModifiers*, the following apply:

TotalSectors

contains the total number of sectors.

BytesPerSector

contains the number of bytes per sector. The IBM OS/2 2.0 File System supports only a value of 512.

Reserved

contains a 0. This alignment field ensures that the GEOMETRY structure aligns with SCSI Read Capacity output.

NumHeads

contains the number of heads.

TotalCylinders

contains the number of cylinders.

SectorsPerTrack

contains the number of sectors per track.

Note: SCSI devices normally do not support cylinder/head/sector (CHS) addressing. However, to maintain INT 13h BIOS compatibility, most controllers create CHS mapping for the devices they support. For non-boot devices, which do not provide INT 13h support, *NumHeads*, *TotalCylinders*, and *SectorsPerTrack* can be set to 0, and the device manager will select appropriate CHS values.

On exit, the driver sets the *Status* and *ErrorCode* fields of the IORBH to reflect the results of the IOCC_GEOMETRY request.

Return Codes

Following is a list of the IOCC_GEOMETRY error codes.

```
IOERR_CMD_NOT_SUPPORTED
IOERR_CMD_SYNTAX
IOERR_CMD_SW_RESOURCE
IOERR_UNIT_NOT_ALLOCATED
IOERR_UNIT_NOT_READY
IOERR_UNIT_PWR_OFF
IOERR_MEDIA_NOT_FORMATTED
IOERR_MEDIA_NOT_SUPPORTED
IOERR_MEDIA_CHANGED
IOERR_MEDIA_NOT_PRESENT
IOERR_ADAPTER_HOSTBUSCHECK
IOERR_ADAPTER_DEVICEBUSCHECK
IOERR_ADAPTER_OVERRUN
IOERR_ADAPTER_UNDERRUN
IOERR_ADAPTER_DIAGFAIL
IOERR_ADAPTER_TIMEOUT
```

IOERR_ADAPTER_DEVICE_TIMEOUT
IOERR_ADAPTER_REQ_NOT_SUPPORTED
IOERR_ADAPTER_REFER_TO_STATUS
IOERR_DEVICE_DEVICEBUSCHECK
IOERR_DEVICE_REQ_NOT_SUPPORTED
IOERR_DEVICE_DIAGFAIL
IOERR_DEVICE_BUSY
IOERR_DEVICE_OVERRUN
IOERR_DEVICE_UNDERRUN

For a detailed description of all the return codes, see Error Handling.

IOCC_EXECUTE_IO

The IOCC_EXECUTE_IO *CommandCode* consists of all *CommandModifiers* responsible for issuing a Read or Write to a unit. The following table describes the IOCC_EXECUTE_IO *CommandModifiers*:

CommandModifier	Description
IOCM_READ	Reads a unit's data into the scatter/gather list buffers.
IOCM_READ_VERIFY	Verifies that the recorded data at the requested I/O address is readable. No data is transferred.
IOCM_READ_PREFETCH	Reads data from the device into the adapter's hardware cache. Support of this command is optional.
IOCM_WRITE	Writes data from the scatter/gather list buffers to the unit's specified I/O address.
IOCM_WRITE_VERIFY	Writes data from the scatter/gather list buffers to the unit's specified I/O address, then verifies that the data can be read (Write/Read Verify combination).

Remarks

Support:	Mandatory
Called By:	OS2DASD.DMD, other device manager, or filter device driver
Context of Call:	TASK, INTERRUPT

Format of IORB

- o IORB Type
 - IORB_EXECUTEIO
- o IORBH Fields

- *CommandCode*
 - o IOCC_EXECUTE_IO
- *CommandModifiers*
 - o IOCM_READ
 - o IOCM_READ_VERIFY
 - o IOCM_READ_PREFETCH
 - o IOCM_WRITE
 - o IOCM_WRITE_VERIFY
- Valid *RequestControl* Flags
 - o IORB_ASYNC_POST
 - o IORB_CHAIN
 - o IORB_CHS_ADDRESSING
(Diskette only, see AF_CHS_ADDRESSING)
 - o IORB_REQ_STATUSBLOCK
 - o IORB_DISABLE_RETRY

IORB_EXECUTEIO Description

This section defines the IORB_EXECUTEIO control block. (See the following table.)

Field Name	C Type	Length	Description
iorbh	IORBH	DB(68)	IORB header
cSGLIST	USHORT	DW	Number of elements
pSGLIST	PSCATGATENTRY	DD	Far pointer to s/g list
ppSGLIST	ULONG	DD	Physical address of s/g list
RBA	ULONG	DD	I/O starting address
BlockCount	USHORT	DW	Sector count
BlocksXferred	USHORT	DW	Number of sectors transferred

BlockSize	USHORT	DW	Number of bytes per sector
Flags	USHORT	DW	I/O-specific flags

On entry to the driver:

iorbh

See IORB General Format.

cSGList

contains the number of scatter/gather elements in the scatter/gather list (*pSGLIST*).

pSGLIST

contains a far pointer to the scatter/gather list, supplied by the caller. The scatter/gather list consists of an array of *cSGList* elements, each pointing to a physically contiguous area of real memory in a format defined by the SCATGATENTRY structure. (See the following table.)

Field Name	C Type	Length	Description
ppXferBuf	ULONG	DD	Physical pointer to buffer
XferBufLen	ULONG	DD	Length of buffer

ppSGLIST

contains a 32-bit physical address of the scatter/gather list.

RBA

contains the starting relative block address for the data transfer operation. If the IORB_CHS_ADDRESSING flag is set in the IORBH *RequestControl* field, then the format of the *RBA* field is defined by the CHS_ADDR structure. (See the following table.)

Field Name	C Type	Length	Description
Cylinder	USHORT	DW	Starting cylinder
Head	UCHAR	DB	Starting head
Sector	UCHAR	DB	Starting sector

BlockCount

contains the number of sectors (length = *BlockSize*) to transfer.

Note: If this value exceeds the adapter's maximum transfer size, the driver is responsible for issuing multiple operations to the unit to complete the caller's request.

BlocksXferred

equals 0 on entry. On exit from the driver *BlocksXferred* contains the number of sectors successfully transferred.

BlockSize

contains the number of bytes in a block or sector. The IBM OS/2 2.0 File System supports only a value of 512.

Flags

defines Execute I/O cache control flags, as shown in the table below.

Flag	Description
XIO_DISABLE_HW_WRITE_CACHE	Disable-deferred Write. Indicates the driver should ensure that the requested data is written to the media prior to doing a notification callout.
XIO_DISABLE_HW_READ_CACHE	Disable Read caching. Indicates to the driver that the data being read is of a transient nature and does not need to be retained in the adapter's hardware cache.

Note: The scatter/gather list-related fields (*cSGLIST*, *pSGLIST*, and *ppSGLIST*) are at the same offset as their equivalent pointers in the IOCC_ADAPTER_PASSTHRU and IOCC_FORMAT *CommandCodes*.

On exit, the driver sets the *Status* and *ErrorCode* fields of the IORBH to reflect the results of the IOCC_EXECUTE_IO request.

Return Codes

Following is a list of the IOCC_EXECUTE_IO error codes:

IOERR_CMD_NOT_SUPPORTED
IOERR_CMD_SYNTAX
IOERR_CMD_SGLIST_BAD
IOERR_CMD_SW_RESOURCE
IOERR_CMD_ABORTED
IOERR_UNIT_NOT_ALLOCATED
IOERR_UNIT_NOT_READY
IOERR_UNIT_PWR_OFF
IOERR_RBA_ADDRESSING_ERROR
IOERR_RBA_LIMIT
IOERR_RBA_CRC_ERROR
IOERR_MEDIA_NOT_FORMATTED
IOERR_MEDIA_NOT_SUPPORTED
IOERR_MEDIA_WRITE_PROTECT
IOERR_MEDIA_CHANGED
IOERR_MEDIA_NOT_PRESENT
IOERR_ADAPTER_HOSTBUSCHECK
IOERR_ADAPTER_DEVICEBUSCHECK
IOERR_ADAPTER_OVERRUN
IOERR_ADAPTER_UNDERRUN
IOERR_ADAPTER_DIAGFAIL
IOERR_ADAPTER_TIMEOUT
IOERR_ADAPTER_DEVICE_TIMEOUT
IOERR_ADAPTER_REQ_NOT_SUPPORTED
IOERR_ADAPTER_REFER_TO_STATUS
IOERR_DEVICE_DEVICEBUSCHECK
IOERR_DEVICE_REQ_NOT_SUPPORTED
IOERR_DEVICE_DIAGFAIL
IOERR_DEVICE_BUSY
IOERR_DEVICE_OVERRUN
IOERR_DEVICE_UNDERRUN

For a detailed description of all the return codes, see Error Handling.

IOCC_FORMAT

The IOCC_FORMAT *CommandCode* consists of all *CommandModifiers* responsible for unit format requests. The following table describes the IOCC_FORMAT *CommandModifiers*:

CommandModifier	Description
IOCM_FORMAT_MEDIA	Formats the entire media in the unit. Support of this command is mandatory for SCSI devices that require low-level formatting.
IOCM_FORMAT_TRACK	Formats the specified track on the unit. Support of this command is mandatory for standard diskette media.
IOCM_FORMAT_PROGRESS	Reports the progress of the formatting. Support of this command is mandatory for standard diskette media.

Remarks

Support: See *CommandModifiers*.
Called By: OS2DASD.DMD, other device manager, or filter device driver
Context of Call: TASK, INTERRUPT

Format of IORB

- o IORB Type
 - IORB_FORMAT
- o IORBH Fields
 - *RequestControl*: Flags can be enabled or disabled.
 - *CommandCode*
 - o IOCC_FORMAT
 - *CommandModifiers*

- o IOCM_FORMAT_MEDIA
 - o IOCM_FORMAT_TRACK
 - o IOCM_FORMAT_PROGRESS
- Valid *RequestControl* Flags
- o IORB_ASYNC_POST
 - o IORB_CHS_ADDRESSING
(Diskette only, see AF_CHS_ADDRESSING)
 - o IORB_REQ_STATUSBLOCK
 - o IORB_DISABLE_RETRY

IORB_FORMAT Description:

This section defines the IORB_FORMAT control block. (See the following table.)

Field Name	C Type	Length	Description
iorbh	IORBH	DB(68)	IORB header
cSGLIST	USHORT	DW	Number of elements
pSGLIST	PSCATGATENTRY	DD	Far pointer to s/g list
ppSGLIST	ULONG	DD	Physical address of s/g list
FormatCmdLen	USHORT	DW	Length of Format command
pFormatCmd	PBYTE	DD	Pointer to Format command
Reserved_1	UCHAR	DB(8)	Reserved.

On entry to the driver:

iorbh

See IORB General Format.

cSGList

contains the number of scatter/gather elements in the scatter/gather list (*pSGLIST*).

pSGLIST

contains a far pointer to the scatter/gather list supplied by the caller. The scatter/gather list consists of an array of *cSGList* elements, each pointing to a physically contiguous area of real memory in a format defined by the SCATGATENTRY structure. (See the following table.)

Field Name	C Type	Length	Description
ppXferBuf	ULONG	DD	Physical pointer to buffer
XferBufLen	ULONG	DD	Length of buffer

ppSGLIST

contains a 32-bit physical address of the scatter/gather list.

Note: For IOCM_FORMAT_MEDIA, the s/g pointers will point to a *Format Unit Parameter List* as defined by the SCSI-2 specification.

If the SCSI Format Unit CDB does not require a parameter list and other command modifiers, the s/g pointers must be 0.

FormatCmdLen

contains the length of the format command (*pFormatCmd*), in bytes.

pFormatCmd

contains a pointer to device-specific formatting information. For diskette controllers, this points to the FORMAT_CMD_TRACK structure (see the table below). For SCSI devices, this points to a SCSI Format Unit CDB.

Field Name	C Type	Length	Description
Flags	USHORT	DW	Flags
RBA	ULONG	DD	Starting RBA
cTrackEntries	USHORT	DW	Number of track entries

On entry to the driver:

Flags

contains flags to define the request, as shown in the following table.

Flag	Description
FF_VERIFY	Verify after format. If set, this flag indicates that the driver should verify the sectors after formatting.

RBA

contains the starting relative block address for an IOCM_FORMAT_TRACK request. For IOCM_FORMAT_MEDIA and IOCM_FORMAT_PROGRESS requests, this field equals 0. If the IORB_CHS_ADDRESSING flag is set in the IORBH->*RequestControl* field, then the format of the *RBA* field is defined by the CHS_ADDR structure, shown in the following table.

Field Name	C Type	Length	Description
Cylinder	USHORT	DW	Starting cylinder
Head	UCHAR	DB	Starting head
Sector	UCHAR	DB	Starting sector

Note: The scatter/gather list-related fields (*cSGLIST*, *pSGLIST*, and *ppSGLIST*) are at the same offset as its equivalent pointers in the IOCC_EXECUTE_IO and IOCC_ADAPTER_PASSTHRU *CommandCodes*.

On exit, the driver sets the *Status* and *ErrorCode* fields of the IORBH to reflect the results of the IOCC_FORMAT request.

Return Codes

Following is a list of the IOCC_FORMAT error codes:

```
IOERR_CMD_NOT_SUPPORTED
IOERR_CMD_SYNTAX
IOERR_CMD_SW_RESOURCE
IOERR_CMD_ABORTED
IOERR_UNIT_NOT_ALLOCATED
IOERR_UNIT_NOT_READY
IOERR_UNIT_PWR_OFF
IOERR_RBA_ADDRESSING_ERROR
IOERR_RBA_LIMIT
IOERR_RBA_CRC_ERROR
IOERR_MEDIA_NOT_SUPPORTED
IOERR_MEDIA_WRITE_PROTECT
IOERR_MEDIA_CHANGED
```

IOERR_MEDIA_NOT_PRESENT
IOERR_ADAPTER_HOSTBUSCHECK
IOERR_ADAPTER_DEVICEBUSCHECK
IOERR_ADAPTER_OVERRUN
IOERR_ADAPTER_UNDERRUN
IOERR_ADAPTER_DIAGFAIL
IOERR_ADAPTER_TIMEOUT
IOERR_ADAPTER_DEVICE_TIMEOUT
IOERR_ADAPTER_REQ_NOT_SUPPORTED
IOERR_ADAPTER_REFER_TO_STATUS
IOERR_DEVICE_DEVICEBUSCHECK
IOERR_DEVICE_REQ_NOT_SUPPORTED
IOERR_DEVICE_DIAGFAIL
IOERR_DEVICE_BUSY
IOERR_DEVICE_OVERRUN
IOERR_DEVICE_UNDERRUN

For a detailed description of all the return codes, see Error Handling.

IOCC_UNIT_STATUS

The IOCC_UNIT_STATUS *CommandCode* consists of all the *CommandModifiers* responsible for returning a unit's current status. The following table describes these *CommandModifiers*:

CommandModifier	Description
IOCM_GET_UNIT_STATUS	Returns flags indicating the unit's current <i>Ready</i> , <i>Power On</i> , and <i>Defective</i> status. For SCSI devices, if a SCSI Target is detected, the driver must issue a SCSI <i>Test Unit Ready</i> command to obtain the current unit status.
IOCM_GET_CHANGELINE_STATE	Returns the unit's current changeline state. This command is mandatory for standard diskette devices.
IOCM_GET_MEDIA_SENSE	Returns the unit's current media storage capacity. This command is mandatory for standard diskette devices.
IOCM_GET_LOCK_STATUS	Returns media sense information This command is mandatory for standard diskette devices.

Remarks

Support: Mandatory (See *CommandModifiers* for exceptions.)
Called By: OS2DASD.DMD, other device manager, or filter device driver
Context of Call: TASK, INTERRUPT

Format of IORB

- o IORB Type

- IORB_UNIT_STATUS
- o IORBH Fields
 - *CommandCode*
 - o IOCC_UNIT_STATUS
 - *CommandModifiers*
 - o IOCM_GET_UNIT_STATUS
 - o IOCM_GET_CHANGELINE_STATE
 - o IOCM_GET_MEDIA_SENSE
 - o IOCM_GET_LOCK_STATUS
- Valid *RequestControl* Flags
 - o IORB_ASYNC_POST
 - o IORB_REQ_STATUSBLOCK

IORB_UNIT_STATUS Description

This section defines the IORB_UNIT_STATUS control block.

IORB_UNIT_STATUS

Field Name	C-Type	Length	Description
iorbh	IORBH	DB(68)	IORB header
UnitStatus	USHORT	DW	Unit status

On entry to the driver:

iorbh

See IORB General Format.

UnitStatus

equals 0, on entry. On exit from the driver, this field contains the status information request, based on the *CommandModifier* field, as shown in the following table.

CommandModifier	Description
IOCM_GET_UNIT_STATUS US_READY	Unit in Ready state

US_POWER	Unit Powered On
US_DEFECTIVE	Unit Defective
IOCM_GET_CHANGELINE_STATE	
US_CHANGELINE_ACTIVE	Changeline occurred
IOCM_GET_MEDIA_SENSE	
US_MEDIA_144MB	144KB media capacity
US_MEDIA_288MB	288KB media capacity
US_MEDIA_720KB	720KB media capacity
US_MEDIA_UNKNOWN	Media capacity unknown
IOCM_GET_LOCK_STATUS	
US_LOCKED	Unit Locked

Remarks

- o The *UnitStatus* field reports the general status of the unit. If the adapter device driver encounters a SCSI Check condition, it must convert any retrieved sense data into IOERR_* error codes and report these in the IORBH->*ErrorCode* field in addition to setting the *UnitStatus* field.
- o The reporting of units that are powered-off is optional. The driver possibly might be able to determine powered-off units from configuration data stored in non-volatile memory.
- o The detection of defective units is optional.

On exit, the driver sets the *Status* and *ErrorCode* fields of the IORBH to reflect the results of the IOCC_UNIT_STATUS request.

Return Codes

For a detailed description of all the return codes, see IORB General Format.

IOCC_DEVICE_CONTROL

The IOCC_DEVICE_CONTROL *CommandCode* consists of all the *CommandModifiers* responsible for device control.

The following table describes the IOCC_DEVICE_CONTROL *CommandModifiers*:

CommandModifier	Description
IOCM_ABORT	Aborts the unit's current operation and causes the driver to return any pending work in its queues. Support is mandatory for SCSI devices.
IOCM_RESET	Resets the unit to its default operating parameters. Support is mandatory for SCSI devices.
IOCM_SUSPEND	Suspends the unit's current operation. This command provides for sharing disk controller hardware with other device drivers. Support is mandatory for diskette controllers.
IOCM_RESUME	Resumes the unit's suspended operation. This command provides for the sharing of the diskette controller with other device drivers. Support is mandatory for diskette controllers.
IOCM_LOCK_MEDIA	Locks the current media in the unit. Support is mandatory for SCSI adapter device drivers and for other devices that support a media-locking function.

IOCM_UNLOCK_MEDIA	<p>Unlocks the current media from the unit.</p> <p>Mandatory for SCSI adapter device drivers and for other devices that support a media-locking function.</p>
IOCM_EJECT_MEDIA	<p>Ejects the current media from the unit.</p> <p>Mandatory for SCSI adapter device drivers and for other devices that support a media-locking function.</p>

Remarks

Support: See the command descriptions.
 Called By: OS2DASD.DMD, other device manager, or filter device driver
 Context of Call: TASK, INTERRUPT

Format of IORB

- o IORB Type
 - IORB_DEVICE_CONTROL
- o IORBH Fields
 - *CommandCode*
 - o IOCC_DEVICE_CONTROL
 - *CommandModifiers*
 - o IOCM_ABORT
 - o IOCM_RESET
 - o IOCM_SUSPEND
 - o IOCM_RESUME
 - o IOCM_LOCK_MEDIA
 - o IOCM_UNLOCK_MEDIA
 - o IOCM_EJECT_MEDIA
 - Valid *RequestControl* Flags
 - o IORB_ASYNC_POST
 - o IORB_REQ_STATUSBLOCK
 - o IORB_DISABLE_RETRY

IOCB_DEVICE_CONTROL Description

This section defines the IORB_DEVICE_CONTROL control block. (See the following

table.)

Field Name	C-Type	Length	Description
iorbh	IORBH	DB(68)	IORB header
Flags	USHORT	DW	Flags
Reserved	USHORT	DW	Reserved

On entry to the driver:

iorbh

See IORB General Format.

Flags

contains flags defined only for IOCM_SUSPEND requests. For all other requests, this field equals 0.

The following table describes the IOCM_SUSPEND flags.

Flag	Description
DC_SUSPEND_DEFERRED	Suspend on idle. If set, this flag indicates that the suspend should occur once the unit is idle.
DC_SUSPEND_IMMEDIATE	Suspend immediate. If set this flag indicates that the suspend should occur once the current request is complete.

Reserved

contains a 0.

On exit, the driver sets the *Status* and *ErrorCode* fields of the IORBH to reflect the results of the IOCC_DEVICE_CONTROL request.

Return Codes

Following is a list of the IOCC_DEVICE_CONTROL error codes:

IOERR_CMD_NOT_SUPPORTED
IOERR_CMD_SYNTAX

IOERR_CMD_SW_RESOURCE
IOERR_UNIT_NOT_ALLOCATED
IOERR_UNIT_NOT_READY
IOERR_UNIT_PWR_OFF

For a detailed description of all the return codes, see Error Handling.

IOCC_ADAPTER_PASSTHRU

The IOCC_ADAPTER_PASSTHRU *CommandCode* consists of all the *CommandModifiers* responsible for issuing SCSI-formatted requests to a unit. The following table describes the *CommandModifiers*:

CommandModifier	Description
IOCM_EXECUTE_SCB	Issues an IBM Subsystem Control Block (SCB) request to the specified unit. Support of this command is optional.
IOCM_EXECUTE_CDB	Issues a SCSI Command Descriptor Block (CDB) request to the specified unit. This command is mandatory for all SCSI units.

Remarks

Support: Mandatory for SCSI units
(See *CommandModifiers* for exceptions.)
Called By: OS2DASD.DMD, other device manager, or filter device driver
Context of Call: TASK, INTERRUPT

Format of IORB

- o IORB Type
 - IORB_ADAPTER_PASSTHRU
- o IORBH Fields
 - *CommandCode*
 - o IOCC_ADAPTER_PASSTHRU
 - *CommandModifiers*
 - o IOCM_EXECUTE_SCB
 - o IOCM_EXECUTE_CDB
 - Valid *RequestControl* Flags

- o IORB_ASYNC_POST
- o IORB_REQ_STATUSBLOCK
- o IORB_DISABLE_RETRY

IORB_ADAPTER_PASSTHRU Description

This section defines the IORB_ADAPTER_PASSTHRU control block. (See the following table.)

Field Name	C-Type	Length	Description
iorbh	IORBH	DB(68)	IORB header
cSGLIST	USHORT	DW	Number of elements
pSGLIST	PSCATGATENTRY	DD	Far pointer to s/g list
ppSGLIST	ULONG	DD	Physical address of s/g list
ControllerCmdLen	USHORT	DW	Length
pControllerCmd	PBYTE	DD	Controller command pointer
ppSCB	ULONG	DD	Physical SCB pointer
Flags	USHORT	DW	Flags

On entry to the driver:

iorbh

See IORB General Format.

cSGList

contains the number of scatter/gather elements in the scatter/gather list (*pSGLIST*), for IOCM_EXECUTE_CDB requests. For all other requests this field contains a 0. **pSGLIST**

contains a far pointer to the scatter/gather list, supplied by the caller, for IOCM_EXECUTE_CDB requests. For all other requests this field contains a 0. The scatter/gather list consists of an array of *cSGList* elements, each pointing to a

physically contiguous area of real memory in a format defined by the SCATGATENTRY structure, shown in the following table.

Field Name	C-Type	Length	Description
ppXferBuf	ULONG	DD	Physical pointer to buffer
XferBufLen	ULONG	DD	Length of buffer

ppSGLIST

contains a 32-bit physical address of the scatter/gather list for IOCM_EXECUTE_CDB requests. For all other requests, this field contains a 0.

ControllerCmdLen

contains the length, in bytes, of the command controller buffer.

pControllerCmd

contains a pointer to the controller command buffer in either SCB or CDB format, based on the *CommandModifier* field.

ppSCB

contains a 32-bit physical address of the Subsystem Control Block (SCB) for IOCM_EXECUTE_SCB requests. For all other requests, this field contains a 0.

Flags

contains flags to define the request, as shown in the following table.

Flag	Description
PT_DIRECTION_IN	Data transfer direction. This flag defines the direction of the data transfer for IOCM_EXECUTE_CDB requests. If set, the data transfer is from the target device to the host adapter. For all other requests, this flag is ignored.

On exit, the driver sets the *Status* and *ErrorCode* fields of the IORBH to reflect the results of the IOCC_ADAPTER_PASSTHRU request.

Return Codes

Following is a list of the IOCC_ADAPTER_PASSTHRU error codes.

IOERR_CMD_NOT_SUPPORTED
IOERR_CMD_SYNTAX
IOERR_CMD_SGLIST_BAD
IOERR_CMD_SW_RESOURCE
IOERR_CMD_ABORTED
IOERR_UNIT_NOT_ALLOCATED
IOERR_UNIT_NOT_READY
IOERR_UNIT_PWR_OFF
IOERR_ADAPTER_HOSTBUSCHECK
IOERR_ADAPTER_DEVICEBUSCHECK
IOERR_ADAPTER_OVERRUN
IOERR_ADAPTER_UNDERRUN
IOERR_ADAPTER_DIAGFAIL
IOERR_ADAPTER_TIMEOUT
IOERR_ADAPTER_DEVICE_TIMEOUT
IOERR_ADAPTER_REQ_NOT_SUPPORTED
IOERR_ADAPTER_REFER_TO_STATUS
IOERR_DEVICE_DEVICEBUSCHECK
IOERR_DEVICE_REQ_NOT_SUPPORTED
IOERR_DEVICE_DIAGFAIL
IOERR_DEVICE_BUSY
IOERR_DEVICE_OVERRUN
IOERR_DEVICE_UNDERRUN

For a detailed description of all the return codes, see Error Handling.

Device Helpers (DevHlp)

In this specification, device helpers are a set of C or assembler callable routines that provide operating system services for OS/2 device drivers. These DevHlp services are RegisterDeviceClass and GetDOSVar.

RegisterDeviceClass

At initialization, the driver calls the *RegisterDeviceClass* DevHlp to register its direct call command handler entry point with the kernel.

Processing

```
LDS  SI, ADD_Name           ; DS:SI = Ptr device driver to ASCIIZ name
                                ;          maximum of 16 chars
MOV  AX, SEGMENT ADD_Function ; AX:BX = Ptr to driver's DirectCall
LEA  BX, ADD_Function       ;          Command Handler
MOV  DI, Device_Flags       ; Must be 0 for adapter device drivers
MOV  CX, Device_Class       ; Must be 1 for adapter device drivers
MOV  DL, DevHlp_RegisterADD
CALL [Device_Help]
```

Results

```
'C' Cleared if successful
AX = ADDHandle
'C' Set if error
AX = ERROR_NOT_ENOUGH_MEMORY
        if CX out of range
        if table is full
```

GetDOSVar

The existing *GetDOSVar* DevHlp has been modified to return a pointer to the device class table for a specified device class. This pointer is valid at initialization, task, and interrupt times, and is used by the device managers or filter device drivers to obtain the adapter device driver entry points from the kernel.

Processing

```
MOV  AL,DHGETDOSV_DEVICECLASSTABLE  ; Device Class table index
MOV  CX,Device_Class                 ; Must be 1 for DISK adapter device driver
MOV  DL,DevHlp_GetDOSVar
CALL [Device_Help]
```

Results

'C' Cleared if successful
AX:BX = global pointer to a table of registered adapter device driver entry points
'C' Set if error

Remarks

- o Adapter Device Driver Entry Point Table format

Note: MAX DCTableEntries can be different for each device class, as follows:

Device_Class = 1 (disk) has a maximum of 32 entries
Device_Class = 2 (mouse) has a maximum of 3 entries


```

struct DevClassTableEntry {
    USHORT    DCOffset;
    USHORT    DCSelector;
    USHORT    DCFlags;
    UCHAR     DCName[16];
};

struct DevClassTableStruc {
    USHORT          DCCount;
    USHORT          DCMaxCount;
    struct DevClassTableEntry  DCTableEntries[MAX];
};

```

Note: The location of the entry point for an adapter device driver can be derived from its adapter device driver handle, as follows:

```

{
    USHORT i = ADDHandle - 1;

    AddSelector = pDevClassTable->DCTableEntries[i].DCSelector ;
    AddOffset    = pDevClassTable->DCTableEntries[i].DCOffset    ;
};

```

Error Handling

To facilitate the use of device managers across a variety of adapter device drivers this specification defines a set of error codes that should be supplied in the *ErrorCode* field of the IORB in the event of a failed operation. The adapter device driver is responsible for translating device error data into these error codes.

Use the following guidelines:

- o Do not program an adapter device driver *defensively*; that is, an adapter device driver should use the services of the device manager and not implement *excessive* safeguards. On the other hand, an adapter device driver must be protected against commands outside of its implemented command set to permit upward compatibility.
- o Program an adapter device driver to protect against timeouts and *hung* devices, transient environmental factors, noise, and so forth.
- o Ensure that the adapter device driver has the capability to properly process any scatter/gather list it receives.
- o Device error information must be translated into the error codes listed in Summary of Error Codes for the OS/2 Device Manager.

Errors must be fully processed by the adapter device driver, as required by the DASD Device Manager. For example, using the IOERR_ADAPTER_REFER_TO_STATUS error code will result in incorrect operation.

- o For other device managers, the same error translation is recommended. If this translation does not produce a reliable error indication, the IOERR_ADAPTER_REFER_TO_STATUS code can be used.
- o An IOERR_RETRY flag is included on commands that must be retried by the adapter device driver. Device managers will ignore this flag because retries must be performed at the adapter device driver level. This flag must be ignored also if the device manager has set the IORB_DISABLE_RETRY bit in the IORB.
- o An IOCM_GET_UNIT_STATUS command is not expected to fail, regardless of the condition of the underlying devices.
- o The IOCM_GET_DEVICE_TABLE command addresses the entire adapter device driver rather than a specific unit; ALLOCATION checks should not be performed.

Summary of Error Codes

This section describes all the adapter device driver error codes. Upon abnormal termination of a direct call command, the adapter device driver sets the `IORB_ERROR` flag in the `IORBH Status` field, and sets the `ErrorCode` field in the `IORBH` with one of the error codes listed below. The error codes are grouped by *error category*. Where stated below, the adapter device driver is required to retry the function prior to returning the error code to the caller.

- o **IOERR_CMD**

This error category maps errors related to the `IORB` command an adapter device driver receives.

IOERR_CMD_NOT_SUPPORTED

This error indicates that the adapter device driver has not implemented the requested function, including commands that the adapter device driver does not *understand*.

IOERR_CMD_SYNTAX

This error indicates that the ADD has detected an inconsistency in the `IORB` that prevents successful execution of the requested function.

IOERR_CMD_SGLIST_BAD

This error indicates that the adapter device driver cannot accept the scatter/gather list passed, due to either a defect in the scatter/gather list (0-length segment) or an underlying hardware limitation.

IOERR_CMD_SW_RESOURCE (*retry required*)

This error indicates that the adapter device driver could not perform the requested function due to the lack of a software resource (for example, buffer, selector, and so forth).

Note: The adapter device driver should attempt to recover from this condition (using a smaller buffer, for example) even if degraded performance results.

IOERR_CMD_ABORTED

This error is returned when an `IOCM_ABORT` is received for a device that is currently processing a command. This error code should be set regardless of SCSI *sense data* that indicates a successful command completion.

IOERR_CMD_ADD_SOFTWARE_FAILURE

This error indicates that the adapter device driver has detected an internal consistency check that prevents it from executing the requested `IORB`.

IOERR_CMD_OS_SOFTWARE_FAILURE

This error indicates that the adapter device driver received an unexpected error return code from an operating system service. The adapter device driver might retry the operation, depending on the nature of the error.

○ **IOERR_UNIT**

This error category maps errors related to the condition of an addressed unit.

IOERR_UNIT_NOT_ALLOCATED

This error indicates that the unit has received an IORB command prior to being allocated. This error should be returned to all commands directed to a unit prior to its receiving an IOCM_ALLOCATE_UNIT command.

IOERR_UNIT_ALLOCATED

This error indicates that an attempt was made to allocate a unit that had been allocated previously. Normally, this error would be returned in response to IOCM_ALLOCATE_UNIT.

IOERR_UNIT_NOT_READY

This error indicates that a unit is unable to perform an otherwise valid operation, usually due to an unusual condition on a unit, such as incorrect spindle speed.

Note: The adapter device driver should not return this error as the result of normal start-up delays on devices.

IOERR_UNIT_PWR_OFF

As an option, the adapter device driver can return this error if it has access to backup configuration data (CMOS) indicating that a previously configured device is not available. If backup configuration data is not available, a powered-off unit normally would not appear in the Adapter Device Driver DEVICETABLE; thus, this error condition would not be possible.

○ **IOERR_RBA**

This error category pertains to problems accessing a relative block address (RBA) on a particular unit.

IOERR_RBA_ADDRESSING_ERROR (*retry required*)

This error indicates that the requested RBA could not be located. This could be due to a failure to find the appropriate address marks on a particular device.

IOERR_RBA_LIMIT

This error indicates that the specified RBA exceeded the allowable maximum for the media currently in the device.

IOERR_RBA_CRC_ERROR (*retry required*)

This error indicates that the RBA was found; however, the data requested could not be read successfully.

- **IOERR_MEDIA**

This error category pertains to problems relating to the media in a drive.

IOERR_MEDIA_NOT_FORMATTED

This error indicates that the requested operation could not be performed since the media in the drive requires low-level formatting. This includes requests to determine media capacity (IOCM_GET_MEDIA_GEOMETRY), if such requests require formatted media.

IOERR_MEDIA_NOT_SUPPORTED

This error indicates that the drive detected media that it cannot support. If the adapter device driver or device cannot make this determination directly, an I/O error can be returned in lieu of this error.

IOERR_MEDIA_WRITE_PROTECT

This error indicates that either the media or the drive is Write protected or that the media is not writable.

IOERR_MEDIA_CHANGED

This error indicates that the media in the drive might have been changed (for example, removal and/or insertion of the media has been detected since the last operation on the device).

IOERR_MEDIA_NOT_PRESENT

This error indicates that the requested operation, requiring media in the drive, failed because media was not in the drive.

- **IOERR_ADAPTER**

This error category pertains to errors that are related to or detected by the host adapter.

IOERR_ADAPTER_HOSTBUSCHECK

This error pertains to problems caused by the adapter's inability to communicate with the host CPU. These errors can include incorrect parity on data received from the host. Frequently these errors are of a severe enough nature to cause shutdown of the host system. In such cases, normal host bus management procedures take precedence over the reporting of this error.

IOERR_ADAPTER_DEVICEBUSCHECK (*retry required*)

This error pertains to problems caused by the adapter's inability to communicate with an attached device. These errors include incorrect parity on data received from the attached device or incorrect bus protocols. These errors are recoverable and should be retried.

IOERR_ADAPTER_OVERRUN (*retry required*)

This error indicates either that the host adapter has lost data from a device due to its buffers being filled faster than they can be emptied

by the host CPU or that a device attempted to supply more data than was expected by the host adapter.

IOERR_ADAPTER_UNDERRUN (*retry required*)

This error indicates either that the host adapter was unable to supply data on demand to a device, which caused device operation to fail, or that a device was expecting more data than the adapter was programmed to supply.

IOERR_ADAPTER_DIAGFAIL

This error indicates that the host adapter detected an internal consistency check, preventing its continued operation. Based on the severity of the error, the adapter device driver may or may not retry the requested operation.

IOERR_ADAPTER_TIMEOUT (*retry required*)

This error indicates that the adapter device driver timeout for an adapter to respond has been exceeded. Normally, the device initiates a retry sequence if this error occurs.

IOERR_ADAPTER_DEVICE_TIMEOUT (*retry required*)

This error indicates the failure of a device to respond to the host adapter.

IOERR_ADAPTER_REQ_NOT_SUPPORTED

This error indicates that the requested operation or function is not supported by this adapter.

IOERR_ADAPTER_REFER_TO_STATUS

This error indicates that the reported error could not be classified. Additional information can be provided in the IORB *StatusBlock* field if requested by the device manager. The ADD should retry the operation unless the IORB_DISABLE_RETRY bit is set.

IOERR_ADAPTER_NONSPECIFIC (*retry required*)

This error should be reported when an ADD cannot classify an adapter-related error condition and an IORB *StatusBlock* value is not provided. If an IORB *StatusBlock* value is provided, IOERR_ADAPTER_REFER_TO_STATUS should be returned.

○ **IOERR_DEVICE**

This error category pertains to errors detected by and relating to devices connected to a host adapter.

IOERR_DEVICE_DEVICEBUSCHECK (*retry required*)

This error pertains to a problem in communications between a host adapter and a device that was detected by the device. This would include incorrect parity on data received by the host adapter or a breakdown in bus protocols between the device and the host adapter.

IOERR_DEVICE_REQ_NOT_SUPPORTED

This error indicates that the requested operation or function is not supported by this device.

IOERR_DEVICE_DIAGFAIL

This error indicates that the device detected an internal consistency check that prevents its correct operation. Depending on the severity of the problem, the ADD might or might not retry the operation.

IOERR_DEVICE_BUSY (*retry required*)

This error indicates that the device is busy and cannot accept the requested operation. This error includes, but is not limited to, SCSI *Contingent Allegiance* conditions.

IOERR_DEVICE_OVERRUN (*retry required*)

This error indicates either that the device has lost data due to its buffers being filled faster than they can be emptied by the host adapter or that the device attempted to supply more data than the host adapter could accept.

IOERR_DEVICE_UNDERRUN (*retry required*)

This error indicates either that the device was unable to obtain data on demand, which caused device operation to fail, or that device operation required more data than was supplied by the host adapter.

IOERR_DEVICE_RESET (*retry required*)

This error indicates that an unexpected device reset occurred that caused device operation to fail. The ADD should retry the failed operation and report this condition as a *recovered error*.

IOERR_DEVICE_NONSPECIFIC (*retry required*)

This error should be returned when the ADD cannot classify a device-related error and an IORB *StatusBlock* value was not supplied. If an IORB *StatusBlock* value is supplied, IOERR_ADAPTER_REFER_TO_STATUS should be returned.

IORB Status Block

The IORB Status Block allows an .ADD driver to provide additional information describing an error condition reported in the IORB ErrorCode field.

Currently, only ADD drivers controlling SCSI devices are required to return a StatusBlock when requested to do so. When an ADD driver does return a StatusBlock, it must set the IORB_REQ_STATUSBLOCK bit in the RequestControl flags to indicate that the StatusBlock has been updated.

SCSI Status Block Format (SCSI_STATUS_BLOCK)

Field Name	C Type	Length	Description
Flags	USHORT	DW	Flags
AdapterErrorCode	USHORT	DW	Adapter related error code
TargetStatus	UCHAR	DB	SCSI Target status
ResidualLength	ULONG	DD	Residual byte count
AdapterDiagInfo	UCHAR	DB(8)	Adapter specific info
ReqSenseLen	USHORT	DW	Request Sense Data allocation
SenseData	PSCSI_REQSENSE_DATA	DD	Request Sense Buffer pointer

Flags

Flags contain bit flags indicating the validity of other fields within the SCSI Status Block.

Flag	Description
STATUS_SENSEDATA_VALID	Set by the ADD driver to indicate that SCSI Sense Data was recovered from the target and placed in the buffer indicated by SenseData.
STATUS_RESIDUAL_VALID	Set by the ADD driver to indicate that the target did not transfer the number of bytes requested. If set the ADD driver is required to return a correct ResidualCount.
STATUS_DIAGINFO_VALID	Set by the ADD driver if it returned adapter specific diagnostic information.
STATUS_DISABLE_REQUEST_SENSE	Set by the client to indicate that the ADD driver must not issue a Request Sense Command to the target regardless of the SCSI status reported.

Note: (SCSI_DISABLE_REQUEST_SENSE 0x0008)

This is a proposed addition to the current ADD/DM specification to simplify the implementation of Device Managers which have clients that explicitly issue their own Request Sense operation. When this bit is set, the ADD driver will not be able to accurately determine an IORB ErrorCode. In this case, the ADD driver must return IOERR_DEVICE_NON_SPECIFIC in the *IORB ErrorCode* field if the target reports other than "GOOD" status.

AdapterErrorCode

An AdapterErrorCode indicates that an adapter related error condition occurred. For example, a SCSI operation which completed successfully but resulted in an adapter detected underrun/overrun would report this condition in this field. The error codes used are defined by the IOERR_* codes in the previous section. If no adapter error condition was detected, then this field must be cleared by the ADD driver.

Note: It is possible for a SCSI operation to fail, but this field would be zero. This would be the usual case for target generated error conditions.

Conversely, it is possible for a SCSI operation to succeed at the IORB level, for example, no IORB Error code reported, but this field would non-zero. This would be the case when the SCSI adapter detected an overrun/underrun on an otherwise successful SCSI operation.

TargetStatus

TargetStatus indicates the SCSI status byte returned by the target during the SCSI status bus phase.

ResidualLength

ResidualLength indicates the difference between the requested data transfer length in the IORB and the actual number of bytes transferred by the target. This field must always be set to as a non-negative number. The specific error condition Overrun vs Underrun should be indicated by setting the appropriate error code in AdapterErrorCode. If the ADD driver is able to return an accurate ResidualLength, it must set the STATUS_RESIDUAL_VALID bit in the *Flags* field.

AdapterDiagInfo

The AdapterDiagInfo field consists of eight bytes the ADD supplier may use to report vendor specific information to assist in local problem determination. If this information is returned the ADD driver must set the bit STATUS_DIAGINFO_VALID in the Flags field.

ReqSenseLen

The ReqSenseLen field indicates the size of the SenseData buffer available. If the target indicates a SCSI status of CHECK_CONDITION then the ADD driver should issue a SCSI Request_Sense command with a data transfer length indicate by ReqSenseLen.

Note: ADD drivers are required to obtain Sense Data whether or not a Status Block is present. In the absense of a StatusBlock, this would usually be done using internal storage of the ADD driver.

However, if a status block is present, then the ADD driver must transfer no less than the number of bytes indicated by the *ReqSenseLen* field and provide this data in the SenseData buffer.

SenseData

The `SenseData` field points to a data buffer to receive `SenseData` returned by the SCSI target as a result of a `REQUEST SENSE` operation issued by the `ADD` driver.

Adapter Device Driver Command-Line Parameters

Following is a diagram of an adapter device driver command-line structure:

```
BASEDEV=AddName.ADD
```

```
Driver-Parameters    Adapter-Parameters    Unit-Parameters
```

Syntax Conventions

Following are the adapter device driver syntax conventions:

- o Command-line contents are case-insensitive.
- o All command-line options begin with the slash character (/).
- o The exclamation character (!) is a negation operator; that is, it negates the option that follows it. The colon character (:) indicates that a list of one or more unit IDs follows the option.
- o The alphabetic *d* character (<d>) indicates a decimal digit.
- o The alphabetic *h* character (<h>) indicates a hexadecimal digit.

Command-Line Parameter Classes

An adapter device driver command line contains three classes of parameters:

- **Adapter Device Driver Parameters**

Adapter device driver parameters apply to all adapters and units managed by an adapter device driver unless overridden by adapter parameters or unit parameters.

- **Adapter Parameters**

Adapter parameters begin with the (/A) switch and identify a specific adapter card. Parameters following the (/A) switch apply only to the adapter card indicated.

- **Unit Parameters**

Unit parameters apply specific units on an adapter.

Note: In some cases, a parameter may appear as both an Adapter parameter and a Unit parameter. If the host adapter hardware supports specifying a parameter on a per-unit basis, then it is recommended that the adapter device driver support both the per-Adapter and per-Unit forms of the parameter.

SCSI-Specific Parameters

The following diagram illustrates a SCSI adapter device driver parameter structure:

```
SCSI-Driver-Parameters ::=
                        /<!--SN
                        /<!--ET
                        /V
```

```
SCSI-Adapter-Parameters ::=
                        /A:d      /S:d      /<!--DM
                        /P:hhhh    /<!--SN
                                    /<!--SN
                                    /<!--ET
                                    /I
```

```
SCSI-Unit-Parameters ::=
                        /<!--DM
                        /<!--SM
                        /<!--SN
                        /<!--ET
                        /<!--HCR
                        /<!--HCW
```

```
SCSI-Target-IDs ::=
                        d              (d=0-7)
                        (d,d)
                        ,
```

Note: All SCSI adapter device drivers must support the following parameters:

```
/V      Verbose
/A      Adapter Identification
/DM     Enable/Disable DASD Manager Support
/SM     Enable/Disable SCSI Manager Support
```

To insure support of various CD-ROM drives the implementation of the following parameters is recommended:

```
/SN     Enable/Disable Synchronous Negotiation
```

/ET Enable/Disable Embedded Target Support

Support of the remaining parameters is optional.

SCSI Adapter Device Driver Parameters

/SN Synchronous Negotiation

This parameter indicates a SCSI Host Adapter should attempt to initiate synchronous data transfers. Negating this parameter (/!SN) indicates that the SCSI Host Adapter must not attempt to initiate synchronous data transfers.

/ET Embedded Targets

This parameter indicates that the adapter device driver must search each SCSI Target for logical units. Negating this parameter (/!ET) indicates that the adapter device driver should only check LUN 0 on each SCSI Target regardless of whether additional Logical Units are present.

/V Verbose

This parameter indicates that the adapter device driver must display diagnostic information during the OS/2 system initialization. The DevHlp_Save_Message device help routine should be used to display this information.

The following format for the displayed information is recommended:

```
XYZ-2010 OS/2 2.0 Driver (yymmdd)
Copyright (c) 1993 XYZ Inc. All Rights Reserved
Adapter: 0 Base Port: 0123  IRQ: 10
  Target: 0 LUN: 0  SCSI_Inquiry_Data (Bytes 8-35)
  Target: 1 LUN: 0  SCSI_Inquiry_Data
  Target: 2 LUN: 0  SCSI_Inquiry_Data
```

/A:d Adapter Index

This parameter specifies the ordering of adapters in the DEVICETABLE returned by the adapter device driver. Normally, adapters are numbered consecutively, starting at 0.

/S:d Adapter Slot ID

For host systems with individually addressable slots, the adapter device driver can designate the location of a host adapter by its slot number, according to the

host system's slot addressing scheme. Typically <d> is a small 0-based number specifying the host system slot.

/P::hhhh Adapter Base I/O Port Address

For host systems with non-addressable slots, the adapter device driver can designate the location of a host adapter by its base I/O port address. Typically, <hhhh> is a 3-4 digit hexadecimal number.

Note: In cases where a specific adapter designation is not made, the adapter device driver can apply its own ordering, based on either the base I/O port address or the physical slot address.

Note: In general, an adapter device driver should choose to support only one of the above addressing methods. If an adapter device driver supports more than one addressing method, it must not permit a mix of addressing methods on a single line.

/DM DASD Manager Support

This parameter indicates that this unit must be supported by the IBM-supplied DASD device manager (OS2DASD.DMD). If this parameter is not specified, the default is to permit DASD device manager support. If this parameter is negated, the adapter device driver must set the UF_NODASD_SUPT flag in the *UnitFlags* field of the DEVICETABLE entry for the device. This parameter is used in conjunction with an OEM-supplied device manager to permit control of specific DASD and SCSI targets.

/SM SCSI Manager Support

This parameter indicates that this unit must be supported by the IBM-supplied SCSI device manager (OS2SCSI.DMD). If this parameter is not specified, the default setting is to permit SCSI device manager support. If this parameter is negated, the adapter device driver must set the UF_NOSCSI_SUPT flag in the *UnitFlags* field of the DEVICETABLE entry for the device. This parameter is used in conjunction with an OEM-supplied device manager to permit control of specific non-DASD and non-SCSI targets.

/I Ignore Adapter

This parameter indicates that adapter device driver should treat the indicated adapter as an uninstalled adapter. The purpose of this parameter is to allow third party software to manage an entire adapter that would normally be managed by the adapter device driver. When specified, the driver must not create a device table entry for the indicated adapter.

/HCW Enable Hardware Write Caching

This parameter is used to control adapter-implemented deferred-write caching for those adapters that support it. If this parameter is not specified, this feature must be enabled. If this parameter is negated, deferred write caching must be disabled on the specified units. Host adapters that do not implement on-board

caching, or that do not have direct control over the operation of the cache, must ignore this parameter if specified.

/HCR Enable Hardware Read Caching

This parameter is used to control adapter-implemented Read caching for those adapters that support it. If this parameter is not specified, this feature must be enabled. If this parameter is negated, Read caching must be disabled on the specified units. Host adapters that do not implement on-board caching, or do not have direct control over the operation of the cache, must ignore this parameter if specified.

d,d... SCSI Embedded Target ID

The above parameters can be followed by a colon (:) with a list of SCSI target IDs, separated by commas. The logical unit number (LUN) for the specified SCSI target is presumed to be 0.

(d,d), (d,d)...SCSI Target/LUN ID

The above parameters can be followed by a colon (:) with a list of SCSI target/LUN pairs in parentheses.

Diskette-Specific Parameters

The following diagram illustrates a diskette-specific adapter device driver parameter structure:

Diskette-Driver-Parameters ::= /MCA

Diskette-Adapter-Parameters ::=
 /A:d
 /DMZ:d
 /IRQ:dd
 /PORT:hhhh

Diskette-Unit-Parameters ::=
 /U:d /<!>AHS
 /F:Drive Capacity
 /SPEC:Specify-Bytes
 /CL:Changeline-Type

Drive Capacity =
 360KB
 1.2MB
 720KB
 1.44MB
 2.88MB

Specify-Bytes = hh, hh

Changeline-Type = /AT
 /PS2
 None

Diskette Adapter Device Driver Parameters

/MCA Install Adapter Device Driver on uChannel machines

This parameter informs the IBM1FLPY adapter device driver to install on uChannel machines. The default is not to install on uChannel machines.

/DMA:d DMA Channel Number

This parameter specifies the DMA channel number that must be used for the diskette adapter. DMA Channel 2 is used if this parameter is not specified.

/IRQ:dd Interrupt Level

This parameter specifies the interrupt level that must be used for the diskette adapter. IRQ 6 is used if this parameter is not specified.

/U:d Unit Number

This parameter specifies the diskette drive number to which options following this parameter apply. Diskette drive numbers start at 0.

Note: To define a third diskette drive on a controller, the /U and /F parameters must be specified.

/AHS Automatic Head Switch

This parameter informs the driver to use a diskette controller feature that automatically switches from Head 0 -> 1. This improves performance by reading both sides of the diskette in a single operation. The default is to enable this option. It may be disabled by negating this parameter.

/F:cccc Drive Capacity

This parameter overrides the BIOS-supplied drive capacity information, enabling the use of drives that the host system's BIOS does not properly recognize. The drive capacity must be suffixed by a (KB) or an (MB).

/SPEC:hh,hhDrive Specify Bytes

This parameter permits the setting of diskette controller *specify bytes*. This is used for drives with unusual or non-standard timing requirements. The correct setting of this parameter varies with drive manufacturers and must be obtained from the drive vendor.

/CL:clt Changeline Type

This parameter permits changing the interpretation of the media change detection signals. The changeline signal can be interpreted according to PC-AT* or PS/2* standards; or checking of the changeline signal can be disabled using this parameter.

ST-506/IDE-Specific Parameters

The following diagram illustrates a ST-506/IDE unit parameter structure:

```
ST-506-Driver-Parameters::=      /V
```

```
ST-506-Adapter-Parameters::=
    /A:d
        /I
        /<!>R
        /IRQ:dd
        /PORT:hhhh
```

```
ST-506-Unit-Parameters::=
    /U:d      /GEO      dd
                (dddd, dddd, dddd)
        /T:dddd
        /SMS
        /LBA
```

ST-506/IDE Adapter Device Driver Parameters

```
/V      Verbose - Display driver information
```

This parameter displays the adapter device driver level, disk controller status and drive geometry information during the OS/2 system initialization.

```
/I      Ignore Adapter
```

This parameter indicates that the IBM1S506 driver should not attempt to initialize the adapter indicated.

This adapter device driver automatically attempts to locate and initialize both the primary and secondary adapters. In some cases other DASD controllers may appear between the primary and secondary IDE controllers. In these cases the system should be configured as follows:

```
BASEDEV=IBMS506.ADD /V /A:1 /I
BASEDEV=MOREDASD.ADD
BASEDEV=IBMS506.ADD /V /A:0 /I
```

/<R Reset Adapter

If this parameter is negated (/!R), adapter resets are disabled. In most cases resets are beneficial to assist in recovering from transient hardware problems such as lost interrupts, timeouts, or commands a particular adapter may not support.

However, for some ESDI adapters, options set by vendor unique commands such as "Sector Sparing" may be lost after a reset. Setting this switch is recommended for ESDI adapters with disks formatted using "Sector Sparing."

/IRQ:dd Interrupt Level

This parameter overrides the default IRQ Number for the adapter indicated. The default IRQ address for Adapter 0 is (14) and for Adapter 1 is (15).

/U:d Unit Number

This parameter specifies the fixed disk drive number to which options following this parameter apply. Fixed disk drive numbers start at 0.

/GEO Drive Geometry

This parameter overrides the Cylinder/Head/Sector geometry for the unit selected. The fourth parameter is the Write Precompensation Cylinder which may be omitted for drives which do not require precompensation.

As an alternate format standard BIOS drive types may be used. Types (0-47) are supported. User defined types 47-49 should be entered directly by in the previous format.

If a second set of geometry is present, then the first set specifies the physical geometry of the drive, and the second set indicates the translated geometry which is reported to the OS/2 system.

/T:dddd Drive Timeout

This parameter indicates the total allowable error recover time for a request. Error recovery times < 5 seconds will be ignored. This parameter defaults to 30 seconds. A shorter interval may be desirable for fault tolerant applications.

/SMS Enable Multiple Block I/O Support

This parameter enables Set Multiple Support which improves performance of some IDE drives. If the drive does not have this feature, this switch will be ignored. The /V - (Verbose) option will indicate whether this feature has been enabled on a particular drive.

/LBA Enable LBA Support

This parameter enables Logical Block Support for IDE drives which support this option. The /V - (Verbose) option will indicate whether this feature has been enabled on a particular drive.

Adapter Presence-Check Services (TESTCFG.SYS)

The TESTCFG device driver provides services for automatic detection of Original Equipment Manufacturer (oem) hardware interfaces. Functions provided by this driver are accessed entirely by opening the device name *TESTCFG\$* and using Category 80h IOctls, as defined below.

The TESTCFG device driver provides:

- o A bus architecture query and some miscellaneous DASD subsystem query functions
- o POS IDs of all installed features for MCA workstations
- o EISA Product IDs of installed features for EISA workstations
- o A copy of the contents of physical memory between locations hex *C0000* and hex *FFFFF*
- o I/O access to all ports not reserved for standard system use

Miscellaneous Queries

TESTCFG provides a query function for determining bus architecture, as follows:

Category: 80h

Function: 60h

Parameter packet:

```
ULONG command;           // Select query function:
                          //    0 = Query bus architecture
```

Data packet: On return, filled in as follows: code.

Query function 0 (bus architecture):

```
ULONG   bus_arch;        // 0 = ISA
                          // 1 = Micro Channel
                          // 2 = EISA
```

Remarks:

None.

Returning All POS IDs

TESTCFG provides a list of POS IDs in a Micro Channel bus system, as follows:

Category: 80h

Function: 61h

Parameter packet:

ULONG command; // Must be zero.

Data packet: On return, filled in as follows:

```
USHORT pos_id[16];            // Start of array of POS IDs returned.  
                              // pos_id[1] == POS ID in slot #1.
```

Remarks:

Returns pos_id[n]=0 in an ISA or EISA configuration.

Returning All EISA IDs

TESTCFG provides a list of EISA IDs in each of the slots of an EISA bus system, as follows:

Category: 80h

Function: 62h

Parameter packet:

ULONG command; // Must be zero.

Data packet: On return, filled in as shown here.

```
UCHAR   product_id[16][4];  
                                // Start of array of EISA IDs returned.  
                                // id[1] == Product ID in slot #1.
```

Remarks:

Returns all product_id[] = 0 in an ISA or Micro Channel configuration.

Obtaining a Copy of BIOS/Adapter Memory

TESTCFG provides a copy of the contents of adapter memory, as follows:

Category: 80h

Function: 40h

Parameter packet:

```
ULONG    command;           // Must be zero.
ULONG    addr0;             // Interpreted as a physical address.
                               // Must be in range of hex C0000 to hex FFFFF.
USHORT   nbytes;            // Number of bytes of memory to copy.
```

Data packet: On return, filled in as shown here.

```
BYTE     content[];         // Contents of memory at specified location.
```

Remarks:

The contents of memory will not be copied past the physical address hex *FFFFFF*.

Issuing an "IN" I/O Instruction

TESTCFG issues an *IN* I/O instruction, as follows:

Category: 80h

Function: 41h

Parameter packet:

```
USHORT  io_address;      // I/O address
USHORT  data_width;      // Integer; # bytes in transfer value.
                        // 1 - in Byte
                        // 2 - in Word
                        // 4 - in DWord
```

Data packet:

```
ULONG   value;           // Value read.
```

Remarks:

Ports below address hex *100* are not accessible. `ERROR_INVALID_PARAMETER` is returned when such a port is referenced.

Issuing an "OUT" I/O INSTRUCTION

TESTCFG issues an *OUT* I/O instruction, as follows:

Category: 80h

Function: 42h

Parameter packet:

```
USHORT  io_address;      // I/O address
USHORT  data_width;      // Integer; # bytes in transfer value.
                                // 1 - Out Byte
                                // 2 - Out Word
                                // 4 - Out DWord
ULONG    value;          // Data value to write.
```

Data packet: None.

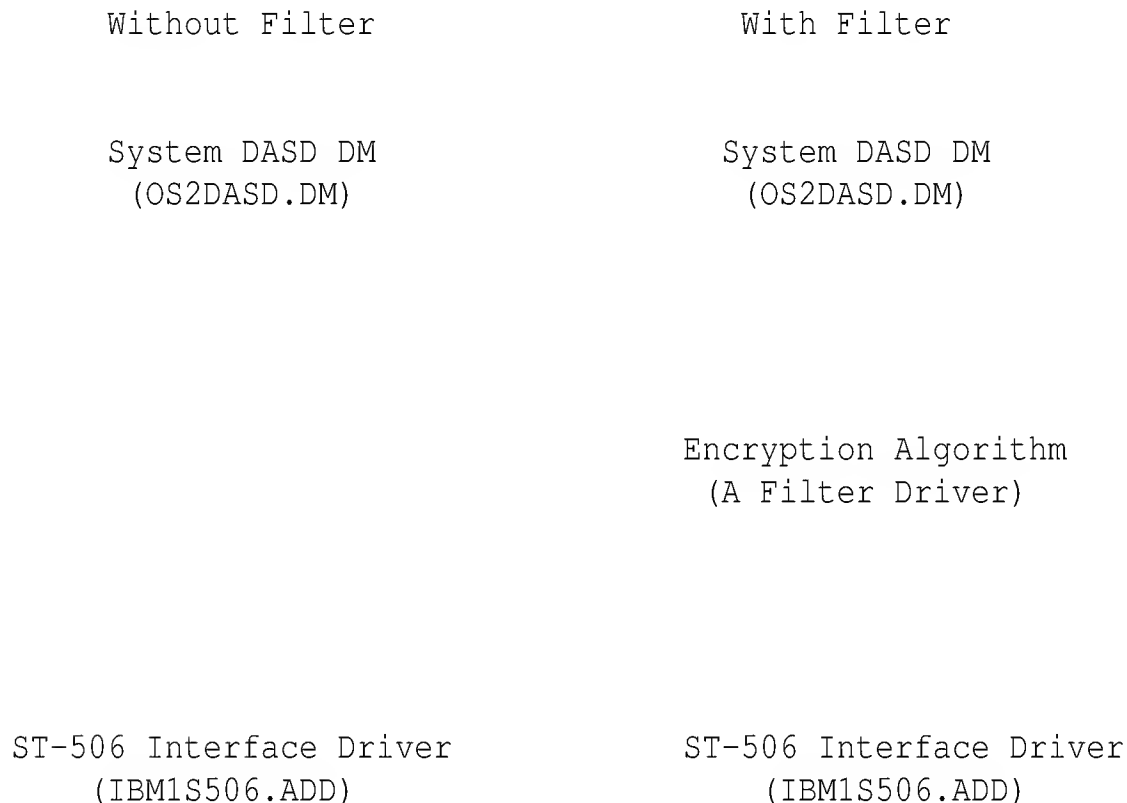
Remarks:

Ports below address hex *100* are not accessible. `ERROR_INVALID_PARAMETER` is returned when such a port is referenced.

Using Filter Device Drivers

There are a number of scenarios in which it is useful to insert one or more filtering algorithms between a device manager and the adapter device driver that is driving the device interface. This is accomplished under the adapter device driver model by installing one or more *filter device drivers* into the call-down path between the DM and the device-interfacing adapter device driver. Filter device drivers are also referred to as filter adapter device drivers, filter drivers, or simply filters.

A sample scenario that utilizes a filter device driver to encrypt the data maintained on a DASD unit is depicted in the following figure:



Filter algorithms are packaged as filter device driver drivers and, in general, they provide the same set of services as any other adapter device driver. Once initialized, filter device drivers receive IORBs from upstream drivers (for example, device managers), perform the filtering function on the data in the IORB, then pass the IORB to call down to the adapter device drivers that the filter device driver is controlling.

One or more filter device drivers can be inserted into the call-down path for a selected device.

One or more call-down paths can share the same filter device driver. For example, multiple call-down paths can share a filter device driver that is providing an encryption function.

The remainder of this chapter contains detailed information on how filter device drivers can be constructed and, subsequently, inserted into the device support for a given I/O system.

Strategies for Providing Filter Functions

There are two strategies for inserting a filter device driver into the call-down path for a given unit's device support:

1. Edit the target unit's UNITINFO table, but do not allocate permanent ownership of the unit.
2. Allocate the target unit, and present a new UNITINFO table to any upstream driver that might issue I/O requests.

In most cases, the first strategy, in which the caller does not permanently allocate the unit, is simpler than the second. The filter device driver simply daisy-chains a filter indicator into the UNITINFO structure of the target unit; then, I/O that otherwise would go directly to the target unit's adapter device driver is redirected through the filter device driver.

The second strategy is required when the filter device driver needs to hide units. For example, a data-stripping feature can be implemented using a filter device driver as follows. The data-stripping filter device driver must allocate all target units to hide them from upstream device managers. Then the data-stripping filter device driver constructs a new UNITINFO table to contain the appropriate information for presenting a logical view of a single, logical (stripped) drive.

Installation and Initialization

Filter device drivers are installed the same as adapter device drivers, using BASEDEV= statements in the CONFIG.SYS file of the workstation. In CONFIG.SYS, the filter device driver is loaded *after* any adapter device drivers it will control but *before* any device managers that the filter device driver will serve; this is ensured by use of the FLT file-name extension.

When the filter device driver receives its initialization packet from the kernel, it must scan the workstation's configuration to determine which units it wants to control, just as a device manager must when it initializes. A filter device driver uses the DevHlp_GetDOSVar to obtain a list of the entry points for all installed adapter device drivers, then it calls each ADD to obtain their device tables. The filter device driver must provide storage for these device tables.

Once the device tables are obtained, each is scanned by the filter device driver for units of interest. Having located the units of interest, the filter device driver must take one of the two actions previously listed, depending on whether the filter driver is using the permanent allocation method.

Editing an Adapter Device Driver Device Table

If the filter device driver does not need to hide the downstream units, it can initiate filtering operations by the following steps.

1. Change the value of the *FilterADDHandle* field in the target unit's UNITINFO structure so that the field selects the filter device driver.

When no filter device drivers are installed, the *FilterADDHandle* value will be 0. So, when a device manager (or other upstream adapter device driver) finds a 0 value in this field, the referenced adapter device driver is directly managing the device interface.

2. Change the *UnitHandle* field of the target unit's UNITINFO structure to a value assigned by the filter device driver.

Notice that the filter device driver is daisy-chaining itself into the call-down path for a given unit. As a result, the filter device driver must save the existing values in *FilterADDHandle* (if nonzero) and *UnitHandle* for the downstream driver. After the filter device driver processes a service request, it must pass the request to the downstream filter device driver or device-interface adapter driver.

The following protocol must be adhered to when editing a UNITINFO structure of another adapter device driver.

The filter device driver alters the information provided in the target UNITINFO structure by using the (IOCC_UNIT_CONTROL) IOCM_CHANGE_UNITINFO command. To issue IOCM_CHANGE_UNITINFO, the filter device driver first must allocate the unit, change the UNITINFO information, and then deallocate the unit.

Changing the UNITINFO information does not affect the operation of the downstream adapter device driver. For example, if a filter device driver changes the UF_HW_SCATGAT bit, the downstream device driver's treatment of the unit is not affected. However, the downstream adapter device driver must present the changed UNITINFO structure when its DEVICETABLE is requested. It is the responsibility of the filter device driver to convert the changed unit definition it sets to the actual unit definition of the adapter device driver owning the unit.

A filter device driver can modify a unit's flags without actually *hooking* the unit. For example a filter device driver could UF_set the A_DRIVE flag without actually receiving requests by leaving the original *UnitHandle* and *FilterADDHandle* fields intact.

Allocating Permanent Ownership of a Unit

Alternatively, a filter device driver can allocate permanent ownership of the target unit from the downstream driver and present a device table containing the new representation of the unit to any upstream drivers. Since the filter device driver retains ownership of the downstream resource, it is not necessary to edit to the downstream driver's UNITINFO structures.

IORBs and Filtering

Once installed, a filter device driver can apply the following to the IORBs it is filtering:

- o Generally, the filter device driver will retain the original IORB and create new IORBs to pass on to the downstream drivers.
- o However, a filter device driver can modify an IORB it receives and pass on the same copy of the IORB data structure (as opposed to passing on a local copy of the IORB). If the adapter device driver does this, it must alter the notification address and restore any input fields it had modified prior to doing notification callouts back to the upstream driver.

The filter device driver must not assume that the contents of the *pIORB->ADDWorkspace* field will be preserved by a downstream driver.

Library and Services

A complement of library services for common adapter device driver tasks is provided in the *IBM Device Driver Source Kit For OS/2*. This adapter device driver library includes a set of functions that can be statically linked with an adapter device driver at build time.

These library services are provided in both source and object form. This code is in the \addcalls and \devhelp subdirectories of the \src tree. You can modify and extend this code to suit your needs.

The *DevHlp* services are provided with FAR code and data-calling convention support. *Adapter Device Driver Calls* services are generally provided with both FAR and NEAR calling-convention support.

The library services include the following:

- o 'C' interface to the DevHlp kernel services
- o Timer services
- o Scatter/gather buffer transfers
- o RBA <-> CHS computations
- o DMA setup and channel control, ISA bus machines
- o Command line parsing

See the headers of the individual functions for a detailed description of function services and their calling conventions.

Command-Line Parsing

To facilitate parsing of command-line parameters and to help encourage uniformity in command-line syntax, a parser/tokenizer is provided in the *IBM Device Driver Source Kit For OS/2*. See Adapter Device Driver Command-Line Parameters for a command-line syntax definition.

The output of the parser/tokenizer is a stream of tokens that represent the contents of the command line. The parser/tokenizer performs preliminary syntactical checks on the command line and indicates the results of these checks in return codes.

As with the other library services provided in the *IBM Device Driver Source Kit For OS/2*, you can modify the parser and its included tables to add adapter-unique flags and parameters.

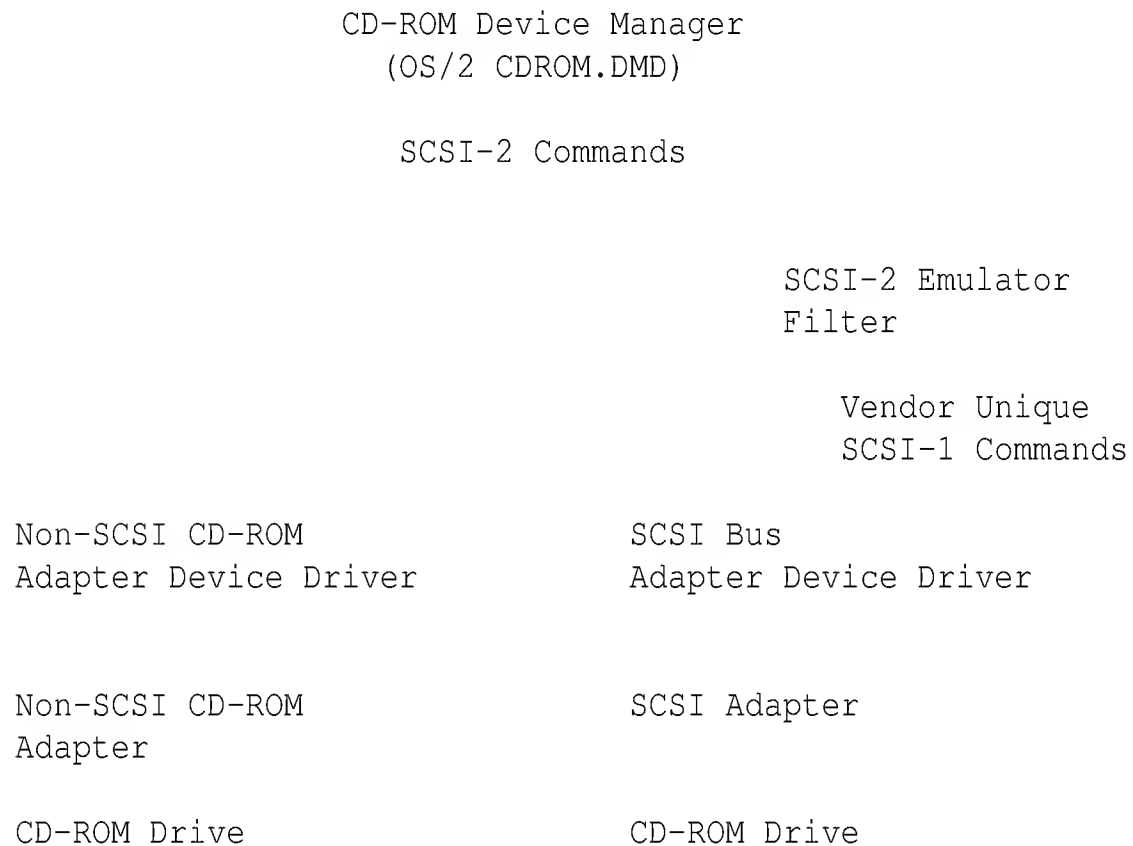
CD-ROM Device Manager Interface Specification

This chapter contains a description of:

- o CD-ROM device management
- o SCSI and Non-SCSI adapter drivers
- o Command support

Overview

The following figure illustrates the layered CD-ROM Device Management structure in the OS/2 operating system.



The CD-ROM Device Manager

The OS/2 CD-ROM Device Manager (OS2CDROM.DMD) is a generic CD-ROM device driver for CD-ROM drives that comply with the ANSI SCSI-2 standard X3T9.2/86-109 (SCSI-2 draft proposed American National Standard Revision 10g). The device driver provides generic data and audio support for drives that support the command set specified in that standard. Vendor unique CD-ROM XA support and multisession Photo CD support is provided for selected drive models.

The CD-ROM Device Manager provides a uniform interface between its clients and adapter device drivers. Clients of the Device Manager include:

- o CD-ROM Installable File System (CDFS.IFS)
- o Multimedia Presentation Manager/2* subsystem
- o virtual CD-ROM device driver (VCDROM.SYS)
- o OS/2 applications

The CD-ROM File System communicates with the CD-ROM Device Manager using the request packet interface defined in the *OS/2 Physical Device Driver Reference* version 2.00 or later.

The Multimedia Presentation Manager/2 subsystem and OS/2 applications communicate with the CD-ROM Device Manager using the Category 80 and 81 IOCTL services defined in the *OS/2 Physical Device Driver Reference*.

DOS applications communicate with the CD-ROM Device Manager indirectly through the virtual DOS CD-ROM device driver VCDROM.SYS. VCDROM.SYS provides virtual MSCDEX support and converts the DOS application request into a file system or Cat 80/81 IOCTL request which is routed to the CD-ROM Device Manager using system device helper services.

The interface between the CD-ROM Device Manager and adapter device drivers adheres to the adapter device driver interface defined in earlier chapters of this reference. The CD-ROM Device Manager converts a request from its client into a SCSI-2 command descriptor block and routes the SCSI-2 command to the specified adapter device driver. The SCSI-2 commands are sent using the IORB adapter passthru command (command code = ADAPTER_PASSTHRU, command modifier = EXECUTE_CDB).

The CD-ROM Device Manager driver (OS2CDROM.DMD) is an installable block device driver and is loaded using a *DEVICE=* statement in CONFIG.SYS. The driver replaces CDRM.SYS, the CD-ROM device driver shipped in the OS/2 2.1 product.

SCSI-2 Emulation Filters

SCSI CD-ROM target devices with vendor unique commands not supported in the SCSI-2 standard require a SCSI-2 *emulation filter*. The emulation filter maps SCSI-2 commands received from the CD-ROM Device Manager to the vendor unique commands supported by the target device. This support is required to enable audio support on CD-ROM drives that adhere to the SCSI-1 standard. The SCSI-1 standard does not define a standard command set for audio control.

A SCSI-2 emulation filter is required for each vendor unique CD-ROM drive. Typically, a CD-ROM manufacturer uses the same vendor unique command set for all it's CD-ROM drives, therefore, one filter driver is required for each manufacturer.

The filter driver receives SCSI-2 commands from the CD-ROM Device Manager, converts the command to it's vendor unique equivalent, and routes the filtered command to the SCSI adapter device driver. If data returned with the command needs to be filtered, the filter driver regains control when the request is complete, converts the outgoing data to it's SCSI-2 equivalent, and then returns to the CD-ROM Device Manager. The filtering process is transparent to the Device Manager and to the adapter device drivers.

Filter drivers adhere to the adapter device driver interface previously defined. (See DASD, SCSI, and CD-ROM Device Manager Interface Specification.) The filter driver is loaded using the *BASEDEV=* statement in CONFIG.SYS.

SCSI Adapter Device Drivers

A SCSI adapter device driver complies with the adapter device driver interface defined in DASD, SCSI, and CD-ROM Device Manager Interface Specification. It must support the ADAPTER_PASSTHRU command for EXECUTE_CDB requests. SCSI-2 commands are sent from the CD-ROM Device Manager using this command. SCSI sense data must be returned when requested.

Non-SCSI CD-ROM Adapter Device Drivers

Several leading CD-ROM drive manufacturers use a proprietary, non-SCSI, host adapter interface for the CD-ROM drive. To support a non-SCSI CD-ROM drive, an adapter device driver is required that emulates a SCSI-2 target device. This enables the CD-ROM Device Manager to issue a common command set to its target devices, whether or not the firmware on the target device directly supports the SCSI-2 command set.

A non-SCSI CD-ROM adapter device driver adheres to the adapter device driver interface defined in DASD, SCSI, and CD-ROM Device Manager Interface Specification. It must support the ADAPTER_PASSTHRU command for EXECUTE_CDB requests. The SCSI-2 commands are sent from the CD-ROM Device Manager using this command.

The SCSI-2 commands must be emulated by the adapter device driver. This includes sense data which is returned back to the CD-ROM Device Manager.

Non-SCSI CD-ROM Adapter Device Driver Specification

A non-SCSI CD-ROM adapter device driver adheres to the adapter device driver interface defined in DASD, SCSI, and CD-ROM Device Manager Interface Specification. Most commands are received as SCSI-2 command descriptor blocks sent using the IORB ADAPTER_PASSTHRU command. The SCSI command descriptor blocks comply with the ANSI SCSI-2 standard X3T9.2/86-109 (SCSI-2 draft proposed American National Standard Revision 10g).

The following sections describe the mandatory IORB and SCSI-2 commands that a non-SCSI CD-ROM adapter device driver must support. The command structures for the IORB command blocks are defined in IORB Control Blocks.

The C Language definitions for the IORB control blocks are included in I/O Request Block - C Definitions.

Mandatory IORB Command Support

The Adapter Device Driver must support the IORB command set defined in the table below.

Command Code	Command Modifier
IOCC_CONFIGURATION	IOCM_GET_DEVICE_TABLE
IOCC_UNIT_CONTROL	IOCM_ALLOCATE_UNIT IOCM_DEALLOCATE_UNIT IOCM_CHANGE_UNITINFO
IOCC_GEOMETRY	IOCM_GET_MEDIA_GEOMETRY IOCM_GET_DEVICE_GEOMETRY
IOCC_EXECUTE_IO	IOCM_READ
IOCC_UNIT_STATUS	IOCM_GET_UNIT_STATUS IOCM_GET_LOCK_STATUS
IOCC_DEVICE_CONTROL	IOCM_ABORT IOCM_RESET IOCM_LOCK_MEDIA IOCM_UNLOCK_MEDIA IOCM_EJECT_MEDIA
IOCC_ADAPTER_PASSTHRU	IOCM_EXECUTE_CDB

IOCC_CONFIGURATION

For the IOCM_GET_DEVICE_TABLE command, the following information must be returned in the device table.

- o The Adapter-to-Device protocol in the *AdapterDevBus* field of the ADAPTERINFO structure must be set to AI_DEVBUS_NONSCSI_CDROM.
- o The *UnitType* field in the UNITINFO structure must be set to UIB_TYPE_CDROM
- o The UF_REMOVABLE bit must be set in the *UnitFlags* field of the UNITINFO structure. The UF_NODASD_SUPT and UF_NOSCSI_SUPT bits should be set to zero.

The adapter device driver should return all other fields in the device table as specified in DASD, SCSI, and CD-ROM Device Manager Interface Specification.

IOCC_UNIT_CONTROL

The IOCM_ALLOCATE_UNIT, IOCM_DEALLOCATE_UNIT and IOCM_CHANGE_UNITINFO must be supported as specified in DASD, SCSI, and CD-ROM Device Manager Interface Specification.

IOCC_GEOMETRY

The geometry returned must be the same for both the IOCM_GET_MEDIA_GEOMETRY and the IOCM_GET_DEVICE_GEOMETRY commands. For both commands, only the *TotalSectors* and the *BytesPerSector* fields should be set. The *TotalSectors* field should be equal to the last addressable logical block address on the media + 1. This value should correspond to the value returned in the SCSI-2 Read Capacity command + 1, which is the starting address of the lead out area. The *BytesPerSector* field should be set to 2048. The *NumHeads*, *TotalCylinders* and *SectorsPerTrack* fields should be set to 0.

If there is no media present in the drive, the driver should return IOERR_UNIT_NOT_READY.

IOCC_EXECUTE_IO

The adapter device driver must support the IOCM_READ command as specified in DASD, SCSI, and CD-ROM Device Manager Interface Specification.

IOCC_UNIT_STATUS

The adapter device driver must support the IOCM_GET_UNIT_STATUS and IOCM_GET_LOCK_STATUS commands as specified in DASD, SCSI, and CD-ROM Device Manager Interface Specification.

IOCC_DEVICE_CONTROL

The adapter device driver must support the IOCM_ABORT, IOCM_RESET, IOCM_LOCK_MEDIA, IOCM_UNLOCK_MEDIA and IOCM_EJECT_MEDIA commands as specified in DASD, SCSI, and CD-ROM Device Manager Interface Specification.

IOCC_ADAPTER_PASSTHRU

The adapter device driver must support the `IOCM_EXECUTE_CDB` command. The list of mandatory SCSI-2 command descriptor blocks which must be supported via this command is defined in the following section.

Mandatory SCSI-2 Command Support

The following table lists the SCSI-2 commands which must be supported. The command structure for the SCSI-2 command descriptor blocks are defined in the ANSI SCSI-2 standard X3T9.2/86-109 (SCSI-2 draft proposed American National Standard Revision 10g). Developers should refer to the ANSI SCSI-2 specification for a definition of each command. Usage notes for each command, as it relates to implementation under the OS/2 operating system, are included in the following sections.

Command Name	Command Code
Group 0 Commands	
TEST UNIT READY	00h
REQUEST SENSE	03h
READ (6)	08h
SEEK (6)	0Bh
INQUIRY	12h
MODE SELECT (6)	15h (see note)
MODE SENSE (6)	1Ah (see note)
START/STOP UNIT	1Bh
PREVENT/ALLOW MEDIUM REMOVAL	1Eh
Group 1 Commands	
READ CD-ROM CAPACITY	25h
READ (10)	28h
SEEK (10)	2Bh
Group 2 Commands	
READ SUB-CHANNEL	42h
READ TOC	43h
READ HEADER	44h
PLAY AUDIO (10)	45h
PLAY AUDIO MSF	47h
PAUSE/RESUME	4Bh
Vendor Unique Commands	
READ DISK INFORMATION	F0h

Note: The Mode Select and Mode Sense commands need only support the Block Descriptor and the Audio Control Page (Page 0E).

TEST UNIT READY (00h)

The TEST UNIT READY command provides a means to check if the logical unit is ready. Refer to the ANSI SCSI-2 specification for a detailed description of this command.

REQUEST SENSE (03h)

The REQUEST SENSE command requests that the target transfer sense data to the initiator.

The Adapter Device Driver should return 18 bytes of sense data, as shown in the following table.

Bit	7	6	5	4	3	2	1	0
Byte								
0	Valid			Error Code (70h)				
1				Reserved				
2		Reserved				Reserved		
3	(MSB)							
---				Information				
6	(LSB)							
7		Additional Sense Length (0Ah)						
8-11				Reserved				
12			Additional Sense Code					
13		Additional Sense Code Qualifier						
14			Reserved					
15-17			Reserved					

- o A valid bit of zero indicates that the *Information Bytes* field is not defined
- o A valid bit if one indicates the *Information Bytes* field contains valid information

Setting this bit is optional. If set, the *Information Byte* field must contain the logical block address associated with the command in error.

The Sense Key, Additional Sense Code and Additional Sense Code Qualifier represent the error condition and must be returned.

All other sense data fields are unused and should be set to 0.

Refer to the ANSI SCSI-2 specification for a detailed description of this command and the complete list of sense key definitions. See I/O Request Block - C Definitions for a discussion on error processing and the mapping of sense data to IORB error codes.

READ (08h) and READ (028h)

The READ (6-byte) and READ (10-byte) commands request that the target transfer data to the initiator.

The Adapter Device Driver should return with a sense key error of 08h (Blank Check) and additional sense code of 064h (illegal mode for this track) if any of the following events occur.

1. If the requested logical block address is in an audio track and the drive does not support reading raw 2352 byte CD-DA data.
2. If the requested logical block address is a mode 2 sector and the drive does not support reading mode 2 sectors.

Refer to the ANSI SCSI-2 specification for a detailed description of this command.

Reading Mode-2 Sectors

If the CD-ROM drive supports the reading of Mode 2 Form 1 and Mode 2 Form 2 sectors, the Adapter Device Driver should mask the complexity of reading the target sector and return successfully, even if the mode of the target sector does not match the currently specified mode for the drive.

Some drives require the drive be set to the proper mode of the target sector prior to issuing the read. For those drives, the Adapter Device Driver should issue the original read, and if the read fails with an error indicating the current drive mode does not match the mode of the target sector, the adapter device driver must issue the mode select to set the proper mode and then re-issue the read request.

It is the responsibility of the CD-ROM Device Manager to ensure the proper block length is specified via the Mode Select command prior to issuing the read command. So, for Mode 2 Form 1 sectors, the device manager will issue a Mode Select command to set the block length to 2048 bytes. For a Mode 2 Form 2 sector, the device manager will issue a Mode Select to set the block length to 2340 or 2352 bytes, depending on the max block length the drive supports. See the Mode Select command section MODE SELECT (15h).

SEEK (0Bh) and Seek (2Bh)

The SEEK (6 byte) and SEEK (10 byte) commands request that the logical unit seek to the specified logical block address.

The adapter device driver must complete the seek operation successfully even if the target sector is an audio sector. If a seek command to an audio sector is not supported by the drive, the driver should issue an alternative command which can successfully seek to the audio sector.

INQUIRY (12h)

The INQUIRY command requests that information regarding parameters of the target and its attached peripheral devices(s) be sent to the initiator.

The standard inquiry data contains 36 required bytes and should be returned as shown in the following table.

Bit	7	6	5	4	3	2	1	0
Byte								
0	Peripheral Qual. (00h)			Peripheral Device Type (05h)				
1	RMB (1)		Device Type Modifier (00h)					
2	ISO Version (00h)		ECMA Version (00h)			ANSI Version (02h)		
3	Reserved			Response Data Format (02h)				
4	Additional Length (1Fh)							
5	Reserved							
6	Reserved							
7	Reserved							
8	(MSB)							
---	Vendor Identification							
15	(LSB)							
16	(MSB)							
---	Product Identification							
31	(LSB)							
32	(MSB)							
---	Product Revision Level							
35	(LSB)							

- o The Peripheral Qualifier is set to 00h.
- o The Peripheral Device Type is set to 05h (CD-ROM device).
- o The Removable Medium (RMB) bit is set to 1 to indicate the media is removable.
- o The Device Type Modifier is set to 00h.
- o The ISO version is set to 00h.
- o The ECMA version is set to 00h.
- o The *ANSI-Approved Version* field is set to 02h, indicating this driver adheres to the SCSI-2 specification.
- o The *Response Data Format* field is set to 02h to indicate compatibility with the SCSI-2 standard.

The *Vendor Identification* field contains eight bytes of ASCII data identifying the vendor of the product. For example:

Byte	08	09	10	11	12	13	14	15
ASCII	S	O	N	Y	sp	sp	sp	sp
Code	53h	4Fh	4Eh	59h	20h	20h	20h	20h

The *Product Identification* field contains sixteen bytes of ASCII data identifying the product model. For example:

Byte	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
ASCII	C	D	-	R	O	M	sp	C	D	U	5	6	1	sp	sp	sp
Code	43h	44h	2Dh	52h	4Fh	4Dh	20h	43h	44h	55h	35h	36h	31h	20h	20h	20h

The *Product Revision Level* field contains four bytes of ASCII data which indicates the revision level of the controller firmware. For example:

Byte	32	33	34	35
------	----	----	----	----

ASCII	1	.	0	0
Code	31h	2Eh	30h	30h

MODE SELECT (15h)

The Adapter Device Driver must support the Mode Select command for the Mode Parameter Block Descriptor and the Audio Control Parameters Page (0x0E).

The mode parameter list contains a header, followed by zero or more block descriptors, followed by zero or more variable-length pages.

Mode Select Parameter List

The following table describes the mode select header.

Bit	7	6	5	4	3	2	1	0
Byte								
0	Resv.							
1	Medium Type (00h)							
2	Reserved							
3	Block Descriptor Length (00h or 08h)							

The following table describes the mode select block descriptor.

Bit	7	6	5	4	3	2	1	0
Byte								
0	Density Code							
1	(MSB)	Number of Blocks (00h)						

3	(LSB)							
4	Reserved							
5	(MSB)	Block Length						

7								

- o The medium type is set to 00h.
- o The block descriptor length is set to either 00h or 08h.
- o The density code is set to 00h.
- o The number of blocks field is set to 00h. This indicates the entire disk has the block length specified.
- o The block length specifies the length in bytes of each logical block.

Block Length Support

The CD-ROM Device Manager will initialize the block length to 2048 bytes per sector. When the Device Manager receives a Readlong IOCTL request (Category 80h, Function 72h), it will issue a Mode Select to change the block length to either 2340 or 2352 bytes, depending on the maximum block length the drive supports. The CD-ROM Device Manager will only issue the Mode Select command to change the block length if the current block length does not match the requested block length.

Prior to the first Read Long IOCTL command, the CD-ROM Device Manager will issue a set of Mode Select commands with various block length values to determine the maximum block length the drive supports. If the Adapter Device Driver receives a Mode Select command with a Block Length value which is not supported, it should return an error with a Sense Key of 05h (Illegal Request) and the Additional Sense Code set to 26h (Invalid field in parameter list).

If the drive can only read a maximum of 2340 bytes per sector, the CD-ROM Device Manager will append 12 bytes of zeros (where the sync bytes are normally placed) to each sector after the read completes. This will ensure a complete 2352 byte sector is always returned back to the application when a ReadLong IOCTL command is issued.

Audio Control Parameter Page

The audio control parameters page sets the playback modes and output controls for subsequent PLAY AUDIO commands and any current audio playback operation. (See the table below.)

Bit	7	6	5	4	3	2	1	0
Byte								
0	Reserved			Page Code (0Eh)				
1	Parameter Length (0Eh)							
2	Reserved				Immed (1)		Reserved	
3-7	Reserved							
8	Reserved				Output Port 0 Channel Selection			
9	Output Port 0 Volume							
10	Reserved				Output Port 1 Channel Selection			
11	Output Port 1 Volume							
12	Reserved				Output Port 2 Channel Selection (0)			
13	Output Port 2 Volume (0)							
14	Reserved				Output Port 3 Channel Selection (0)			
15	Output Port 3 Volume (0)							

The immediate bit (Immed) is set to 1 to indicate the target shall send completion status as soon as the playback operation has been started.

The Output Port channel selection specifies the audio channels from the disk to which this output port should be connected, as shown in the table below.

Channel Selection	Function
00h	Output port muted.
01h	Connect audio channel 0 (left) to this output port
02h	Connect audio channel 1 (right) to this output port
03h	Connect audio channel 0 and 1 to this output port
04h-0Fh	Not supported

The channel volume control indicates the relative volume level for this audio output port. Values between 0x00 and 0xFF are allowed. A volume level of zero indicates the output is muted, a value of 0xFF indicates maximum volume level.

The Output Port 2 and Output Port 3 channel selection and volume fields (bytes 12 to 15) are reserved and are set to 0.

Audio Control Determination

At initialization time, the CD-ROM Device Manager will issue a series of Mode Select commands with various Audio Control Parameter Pages. The Output Port Channel Select and Volume Level fields will be varied to determine the audio control capabilities of the drive. For example, an Audio Control Parameter Page is sent with volume levels different for Channel 0 and Channel 1 to determine if the drive supports independent volume levels for each channel.

If the Adapter Device Driver receives a Mode Select command with an Audio Control Parameter Page which is not supported by the drive, it should return an error with a Sense Key of 05h (Illegal Request) and the Additional Sense Code set to 26h (Invalid field in parameter list). This allows the CD-ROM Device Manager to determine the audio capabilities of the drive.

MODE SENSE Command (1Ah)

The MODE SENSE (6) command provides a means for a target to report parameters to the initiator. It is a complementary command to the MODE SELECT (6) command. Refer to the ANSI SCSI-2 specification for a detailed description of this command.

START/STOP UNIT Command (1Bh)

The START/STOP UNIT command requests the target enable or disable the logical unit for media access operations. This command is used to eject media from the drive.

If a PREVENT MEDIUM REMOVAL command has been issued, a request to the disk should return with the sense key set to ILLEGAL REQUEST (05h) and the additional sense code set to MEDIUM REMOVAL PREVENT (53h).

Refer to the ANSI SCSI-2 specification for a detailed description of this command.

PREVENT/ALLOW MEDIUM REMOVAL (1Eh)

The PREVENT/ALLOW MEDIUM REMOVAL command requests that the target enable or disable the removal of the medium in the logical unit.

Refer to the ANSI SCSI-2 specification for a detailed description of this command.

READ CD-ROM CAPACITY (25h)

The READ CD-ROM CAPACITY command provides a means for the initiator to request information regarding the capacity of the logical unit. The capacity is based on the starting address of the lead-out area minus one. The logical block address returned is the address of the last user accessible block on the disk.

Refer to the ANSI SCSI-2 specification for a detailed description of this command.

READ SUB-CHANNEL (42h)

The READ SUB-CHANNEL command requests that the target return the requested sub-channel data plus the state of audio play operations.

Note: If a READ SUB CHANNEL command is issued to request the media catalog number (UPC/EAN Bar Coding), the drive should return the UPC code in ASCII format as specified in the SCSI-2 specification. Some drives return the UPC code in BCD. It should be converted to ASCII prior to returning. All other sub-channel data is returned in binary format.

Refer to the ANSI SCSI-2 specification for a detailed description of this command.

READ TOC (43h)

The READ TOC command requests that the target transfer data from the table of contents to the initiator.

For drives which do not support a READ TOC command while an audio play operation is in progress, the adapter device driver should buffer the entire TOC data when media is first mounted in the drive. This will ensure the TOC data is retrievable during an audio play operation.

All TOC data is returned in binary format.

Refer to the ANSI SCSI-2 specification for a detailed description of this command.

READ HEADER (44h)

The READ HEADER command requests that the device return the CD-ROM data block address header of the requested logical block.

Refer to the ANSI SCSI-2 specification for a detailed description of this command.

PLAY AUDIO (10) (45h)

The PLAY AUDIO command requests the target to begin an audio playback operation.

The relative address bit (RelAdr) is not used and will be set to 0.

If the requested starting address is not in an audio track, the Adapter Device Driver should return with a sense key error of 08h (Blank Check) and additional sense code of 064h (illegal mode for this track).

Refer to the ANSI SCSI-2 specification for a detailed description of this command.

PLAY AUDIO MSF (47h)

The PLAY AUDIO MSF command requests the target to begin an audio playback operation.

As specified in the SCSI-2 specification, the starting MSF address and ending MSF address fields are specified in hexadecimal (not in BCD).

If the requested starting address is not in an audio track, the Adapter Device Driver should return with a sense key error of 08h (Blank Check) and additional sense code of 064h (illegal mode for this track).

Refer to the ANSI SCSI-2 specification for a detailed description of this command.

PAUSE/RESUME (4Bh)

The PAUSE RESUME command requests that the device stop or start an audio play operation.

It shall not be considered an error to request a pause when a pause is already in effect or to request a resume when a play operation is in progress.

Refer to the ANSI SCSI-2 specification for a detailed description of this command.

READ DISK INFORMATION (F0h)

Bit	7	6	5	4	3	2	1	0
Byte								
0								Operation Code (F0h)
1								Reserved
2-9								Type

The READ DISK INFORMATION command is a vendor unique command to request information regarding capabilities of the target device. The command is also used to return the starting address of the last session for a multisession photo CD disk.

TYPE = 00b

If the *TYPE* field in the command descriptor block is 00h, the adapter device driver should return the data shown in the following table:

Bit	7	6	5	4	3	2	1	0
Byte								
0		Reserved			CDDA	Form2	Form1	PhotoCD
1				Reserved				
2	(MSB)							
---				Maximum Block Length				
3	(LSB)							

- o A multisession photo CD (PhotoCD) bit set to 1 indicates the drive supports the reading of multisession photo CD disks.
- o A Mode 2 Form 1 (Form1) bit set to 1 indicates the drive supports the reading of Mode 2 Form 1 sectors.
- o A Mode 2 Form 2 (Form2) bit set to 1 indicates the drive supports the reading of Mode 2 Form 2 sectors.
- o A CD-Digital Audio (CDDA) set to 1 indicates the drive supports reading 2352 byte raw CD-DA data.

The *Maximum Block Length* field specifies the maximum block length that can be transferred by the drive during a read operation. The value should match the maximum value which can be specified in the *Block Length* field of the Block Descriptor during a Mode Select command.

TYPE = 01b

If the *TYPE* field in the command descriptor block is 01h, the adapter device driver should return the data shown in the following table:

Bit	7	6	5	4	3	2	1	0
Byte								
0	Reserved							
1	Address of Last Session (Minutes)							
2	Address of Last Session (Seconds)							
3	Address of Last Session (Frame)							

If the installed media is a multisession photo CD disk, the driver should return the absolute address of the last session. The data is returned in MSF format and is expressed in hexadecimal. If the installed media is not a multisession photo CD disk, the driver **must** return zero for all fields in the returned control block.

Multisession Photo CD Support

At initialization time, the CD-ROM Device Manager will issue the READ DISK INFORMATION command with TYPE code = 00h to determine if the drive supports the reading of multisession photo CD disks. If the drive indicates it supports the reading of multisession photo CD disks (by returning with the PhotoCD bit set in the returned READ DISK INFORMATION data block) the CD-ROM Device Manager will issue the READ DISK INFORMATION command with TYPE code = 01h whenever media is first mounted in the drive.

If the adapter device driver indicates that a multisession disk is mounted, the CD-ROM Device Manager will use the returned last session address to map subsequent read requests for Volume Descriptor sectors to the Volume Descriptor sectors in the last session on the disk. With this implementation, the responsibility of mapping sectors is done by the CD-ROM Device Manager and not by each adapter device driver.

Error Processing

When a request is issued, the CD-ROM Device Manager will set the IORB_REQ_STATUSBLOCK bit in the *RequestControl* field of the IORB header. If this bit is set and an error occurs, the the Adapter Device Driver must return a valid status block and a valid sense data block back to the Device Manager when the request completes.

The status block is pointed to by the *pStatusBlock* field in the IORB header. It should be noted that pStatusBlock is a 16 bit near pointer, so the block is within the same segment as the IORB. The sense data is pointed to by the *SenseData* field in the status block. This field is a 16:16 far pointer. The length of the sense data to return is in specified in the *ReqSenseLen* field.

The adapter device driver must set the following fields when returning sense data:

1. The IORB_ERROR and IORB_STATUS_BLOCK_AVAIL bits must be set in the Status field of the IORB header.
2. The STATUS_SENSEDATA_VALID bit must be set in the *Flags* field of the Status Block.
3. The value in the *Error Code* field of the returned sense data must be set to 70h.
4. The Sense Key, Additional Sense Code and Additional Sense Code Qualifier must be set to indicate the returned error.

To maintain consistency with the adapter device driver specification, an IORB error code must also be returned in the IORB header when an error occurs. This is in addition to the returned sense data information. The table below shows the mapping between Sense Data error codes and IORB error codes. If the Adapter Device Driver returns Sense Key and Sense Codes listed in the SCSI-2 specification which are not listed in the table below, the adapter device driver must map the sense key and codes to the most appropriate IORB error code.

Key = Sense Key
ASC = Additional Sense Code
ASCQ = Additional Sense Code Qualifier

Key ASC ASCQ	Description	IORB Error Code
-----------------	-------------	-----------------

NOT	02h 04h 00h	Logical Unit Not Ready	IOERR_UNIT_NOT_READY
READY	02h 04h 01h	Becoming ready	IOERR_UNIT_NOT_READY
	02h 57h 00h	Unable to recover TOC	IOERR_UNIT_NOT_READY
MEDIUM	03h 02h 00h	No Seek Complete	IOERR_RBA_ADDRESSING_ERROR
ERROR	03h 11h 00h	Unrecovered Read Error	IOERR_RBA_CRC_ERROR
	03h 11h 05h	L-EC Uncorrectable Error	IOERR_RBA_CRC_ERROR
	03h 11h 06h	CIRC Unrecovered Error	IOERR_RBA_CRC_ERROR
	03h 12h 00h	Address Mark Not Found	IOERR_RBA_ADDRESSING_ERROR
	03h 15h 00h	Random Positioning Error	IOERR_RBA_ADDRESSING_ERROR
	03h 16h 00h	Data Synchronization Error	IOERR_RBA_CRC_ERROR
	03h 30h 00h	Incompatible Medium	IOERR_MEDIA_NOT_SUPPORTED
HARDWARE	04h 08h 00h	Unit Communication Fail	IOERR_DEVICE_NONSPECIFIC
ERROR	04h 09h 01h	Tracking Servo Failure	IOERR_DEVICE_NONSPECIFIC
	04h 09h 02h	Focus Servo Failure	IOERR_DEVICE_NONSPECIFIC
	04h 09h 03h	Spindle Servo Failure	IOERR_DEVICE_NONSPECIFIC
	04h 44h 00h	Internal Target Failure	IOERR_DEVICE_NONSPECIFIC
ILLEGAL	05h 20h 00h	Invalid Command Code	IOERR_DEVICE_REQ_NOT_SUPPORTED
REQUEST	05h 21h 00h	LBA Out of Range	IOERR_RBA_ADDRESSING_ERROR
	05h 24h 00h	Invalid field in CDB	IOERR_CMD_SYNTAX
	05h 25h 00h	Unit not supported	IOERR_CMD_SYNTAX
	05h 26h 00h	Invalid field in parmlist	IOERR_CMD_SYNTAX
	05h 63h 00h	End of user area	IOERR_RBA_ADDRESSING_ERROR
	05h 64h 00h	Illegal mode for track	IOERR_RBA_ADDRESSING_ERROR
UNIT	06h 28h 00h	Medium may have changed	IOERR_MEDIA_CHANGED
ATTENTION	06h 29h 00h	Power on reset	IOERR_DEVICE_RESET
BLANK	08h 64h 00h	Illegal mode for track	IOERR_RBA_ADDRESSING_ERROR
CHECK			

Building an OS/2 Virtual Disk Driver

This chapter describes how to program and build an OS/2 virtual disk driver. In order to successfully build a virtual disk driver, should be familiar with the OS/2 2.0 operating system or later, and have previous experience developing OS/2 device drivers.

In the *IBM Device Driver Source Kit For OS/2*, you will find an OS/2 virtual disk driver. After reading this chapter and examining the code, you can use this information to write your own virtual device driver.

Overview

The virtual disk driver code provides access to a virtual disk in random access memory. The virtual disk driver runs in a multi-tasking environment and is a protected resource.

In this chapter you will find:

- o A table listing the virtual disk parameters
- o A table listing the virtual disk commands
- o An explanation of how the virtual disk initialization routine works
- o Information for performing time-critical tasks
- o A procedure for building the virtual disk device driver code that is provided with the *IBM Device Driver Source Kit For OS/2*

Using the Virtual Disk Parameters: To allocate the virtual disk driver volume, modify the following device statement in the CONFIG.SYS file.

```
DEVICE = .\PATHNAME\VDISK.SYS [bbbb] [ssss] [dddd]
```

Where:

bbbb Determines the disk size in K bytes. The default value is 64KB. The minimum is 16KB. The maximum is 524 288 (512MB).

ssss Determines the sector size in bytes. The default value is 128. Acceptable values are increments of 128 which include 128, 256, 512, and 1024.

dddd Determines the number of root directory entries. The default is 64; with a minimum of 2 and a maximum of 1024. The value is rounded up to the nearest sector size boundary.

The virtual disk driver adjusts the value of dddd to the nearest sector size boundary. For example if you give a value of 25, and the sector size is 512 bytes, 25 will be rounded up to 32 which is the next multiple of 16.

The parameters you use to specify byte and sector size and the number of directory entries are positional parameters. This means that if you omit a parameter, you should not leave it blank. You should use a comma in the parameter field to separate this field from the next. The only time you can use

blank spaces as separators is in the instance where you are coding blanks for all the parameters.

In the event that there is not enough memory to create the virtual disk driver volume, the driver attempts to create a DOS volume with 16 directory entries. This may result in a volume with a different number of directories than you specified on the device statement (dddd).

To ensure system reliability, specify 32 megabytes or less for disk size.

Example 1

```
C:\OS2\VDISK.SYS ,128,64
```

where the disk size is 64KB, the sector size is 128 bytes, and there are 64 directory entries.

Example 2

```
C:\OS2\VDISK.SYS 2048,,32
```

where the disk size is 2 048 KB, the sector size is 128 bytes, and there are 32 directory entries.

Example 3

```
C:\OS2\VDISK.SYS 2048,512,
```

where the disk size is 2 048 KB, the sector size is 512 bytes, and there are 64

directory entries.

Example 4

```
C:\OS2\VDISK.SYS ,128,32
```

where the disk size is 64 KB, the sector size is 128 bytes, and there are 32 directory entries.

Supported Physical Device Driver Strategy Commands: The virtual disk driver is a block device driver and cannot be partitioned. For this reason, the virtual disk driver uses a limited set of physical device driver strategy commands. These are listed below:

Code	Function
0h	Init
1h	Media Check
2h	Build BPB
4h	Read (Input)
8h	Write (Output)
9h	Write With Verify
Dh	Open Device
Eh	Close Device
Fh	Removable Media
10h	Generic IOCTL
11h	Reset Media
12h	Get Logical Drive Map
13h	Set Logical Drive Map
18h	No Caching (read)
19h	No Caching (write)
1Ah	No Caching (Write With Verify)
1Dh	Get Driver Capabilities

If the virtual disk driver uses any commands other than those shown above, the driver returns an unknown command error code. For more information on these commands, refer to the *OS/2 Physical Device Driver Reference*.

The virtual disk driver supports the Extended Device Driver Interface which is implemented through the Get Driver Capabilities command. This interface issues a Request List of prioritized commands. VDisk_Strat2, specified in the driver capabilities structure, is the entry point for all the commands.

CHKDSK uses the category 08h and function 63h IOCTL command from the kernel. This is the only command supported by the virtual disk driver in the general IOCTL commands category.

Virtual Disk Driver Initialization: The virtual disk driver initialization routine does the following:

- o Initializes various global values and initializes the DevHelp function router address.
- o Parses the command line and sets the values accordingly.

The "DEVICE = xxxxxxxxx" line pointer provided in request packet searches for the various device parameters. The pointer searches through the device name field to obtain the arguments. Then the pointer parses the arguments as they are encountered. All parameter errors are detected at this time. The static initialization routine sets the parameter variables to the default settings.

- o Allocates the memory for the virtual disk driver.

The routine issues the DevHlp_VMalloc command to allocate random access memory for the virtual disk driver.

- o Initializes the DOS volume in random access memory for the virtual disk driver.

To so, the routine sets the BPB and initializes the RESERVED (boot) sector, FAT sectors, and root directory sectors and writes them to the virtual disk driver. First the routine initializes the BPB values. Then the routine writes the BOOT record, containing the BPB, to sector 0. The routine writes to a FAT file with all of the clusters free, and writes to the root directory with ONE entry (the Volume ID at VALID).

- o Prints a report of the RAMDrive parameters.

You can print the BPB values. To do so, use the DosGetMessage and DosPutMessage functions in your virtual disk driver. From this report, you can determine the device size, cluster size, and directory size.

- o Specifies the return INIT I/O packet values.

The INIT I/O packet return values for number of units are set, as well as the BPB array pointer.

At any time during the initialization steps an error may be detected. When this happens, the system prints an error message. The virtual disk driver uninstalls and returns a unit count of 0 in the INIT device I/O packet.

Performing Time-Critical Tasks: To perform time-critical tasks, you must call the DevHlp_GetDOSVar service from the virtual disk driver code. The virtual disk driver periodically checks the TCYield flag and calls the TCYield function to yield the CPU to a time-critical thread. The location of the TCYield flag is obtained from a call to DevHlp_GetDosVar. The virtual disk driver checks the TCYield flag each time 32,768 bytes of data have been transferred. Refer to the *OS/2 Physical Device Driver Reference* for more information.

Building the OS/2 2.1 Virtual Disk Driver sample code: To build the sample virtual disk driver code, complete the following steps:

1. Add the TOOLS directory to the OS/2 *IBM Device Driver Source Kit For OS/2* and set it to the current path.
2. Set the TMP environment variable to point to a work area. This is shown below:

```
SET TMP=E:\
```

3. NMAKE the following makefiles in the DDK:

```
SRC\DEV\VDISK\MAKEFILE  
CD\DDK\SRC\DEV\VDISK  
NMAKE
```

OS2DASD.DMD - Technical Reference

The OS2DASD Device Manager (OS2DASD.DMD) provides the interface between the OS/2 File Systems (FAT, HPFS) and Adapter Drivers (*.ADDs) that support fixed and removable magnetic disks.

This device manager allows the *.ADD drivers to communicate with adapter or disk hardware without interacting with the logical contents of these devices or being affected by the complexity of OS/2 block device driver interfaces. The device manager communicates with *.ADD drivers exclusively by way of Input/Output Request Blocks (IORBs), as described in the preceding chapters of the *Storage Device Driver Reference*.

The primary functions of this device manager are:

1. Implementing the OS/2 Kernel/FileSystem block device driver interfaces; creating IORBs as required for communication to .ADD drivers.
2. Scanning .ADD drivers for removable or fixed magnetic devices.
3. Managing partitioned devices by creating *logical drives* for partitioned disks and reporting these logical drives to the OS/2 kernel.
4. Providing IOCTL interfaces to the file system utility applications to allow for the preparation of the media.
5. Providing for the attachment from a set of logical drives to a physical removable device is called *pseudo drive* support. An example is the mapping of drive A: or B: to a single diskette drive.

Kernel/FileSystem Interfaces

The OS2DASD device manager supports three major types of kernel interfaces: Request Packets, Extended Disk Interface, and Generic IOctls.

With the exception of Generic IOctls, these interfaces are used by the OS/2 Kernel and OS/2 File Systems to communicate with device drivers and are not directly available to applications.

Request Packets

Request Packets are used for small I/O requests or status requests to the device manager.

The following table shows the Request Packets that are supported by the OS2DASD Device Manager:

Cmd Code	Packet Name
01h	Check Media Change
02h	Build BPB
04h	Read
08h	Write
09h	Write with Verify
0Fh	Check Removable
10h	Generic IOCtl
11h	Reset Media Change
12h	Get Logical Drive Map
13h	Set Logical Drive Map
16h	Get Partitionable Disk Count
17h	Map Unit Numbers to Physical Drive
18h	Read (Suppress caching)
19h	Write (Suppress caching)
1Ah	Write w/Verify (Suppress caching)
1Dh	Get Extended Disk Interface Info

The formats of these packets can be found in the *OS/2 Physical Device Driver Reference*. For more information regarding the implementation and interpretation of these packets by OS2DASD, see Request Packet Management.

Extended Disk Interface

The Extended Disk Interface provides a higher performance path for a limited set of commands. The advantages of this interface are:

- o Multiple I/O requests may be submitted in a single list.
- o Each request may transfer data to or from discontinuous areas of memory.
- o Each request may specify a priority to other I/O requests.
- o A file system can directly call the device driver to submit I/O requests, rather than going through the OS/2 kernel.
- o This interface may be used at interrupt time.

See Request Lists and Request Control for information about the format of the Extended Disk Interface "Request Lists".

Generic IOctls

IOctl interfaces are generally used by the file system utility applications such as FORMAT and CHKDSK to prepare or access media when a file system is not operating. The interfaces also perform operations that query or change hardware-specific characteristics of a device.

Category 08h IOctls apply to a single drive letter or partition. Most file system utility programs access disks using this IOctl category.

Category 09h IOctls apply to the entire physical device. In other words, these IOctls ignore any partitioning scheme that may be present on the drive. Partitioning utility programs such as FDISK and FDISKPM use this IOctl category to set up a disk partitioning scheme.

The OS2DASD Device Manager supports the IOctls shown in the following table:

Category	Function	Purpose
08h	22h	Create Alias Drive Letter
08h	40h	Lock/Unlock/Eject Media
08h	43h	Set Drive Parameters
08h	44h	Write Track
08h	45h	Format/Verify Track, Multitrack Verify
08h	5Dh	Stop/Start Diskette Controller
08h	60h	Read Diskette Media Type Switches
08h	63h	Get Drive Parameters
08h	64h	Read Track
08h	65h	Verify Track
08h	66h	Get Drive Status - Locked/Unlocked/Ready
09h	44h	Physical Volume - Write Track

09h	63h	Physical Volume - Get Drive Parameters
09h	64h	Physical Volume - Read Track
09h	65h	Physical Volume - Verify Track

Block Device Management

When the OS2DASD initializes, it scans .ADD drivers for fixed or removable magnetic devices. For each device found, the driver creates an internal control block that is called a *UnitCB*.

To access the device, the UnitCB provides the linkage to the corresponding ADD driver and ADD UnitHandle.

When UnitCBs have been created, the driver creates VolCBs to represent each of the following:

- o Physical non-removable drive
- o Removable drive
- o Logical volumes on a partitioned drive

VolCBs are linked together to create a unit number ordering system, based on DOS conventions. In addition, VolCBs are linked to their corresponding UnitCB, which provides the information necessary to access the physical device by way of the ADD drivers.

Unit numbers are *not* equivalent to drive letters. The OS/2 Kernel/FileSystems assigns drive letters. For example, a block device driver cannot demand that a particular set of drive letters be assigned to it.

OS2DASD assigns unit numbers as follows:

80h - 98h Physical Drives

A VolCB is created for each non-removable drive found. This VolCB represents the drive as a single device and ignores any partitioning scheme. For each drive found, a unit number is assigned from 80h to 98h.

0,1 Reserved for diskette drives

Regardless of whether an .ADD driver is found claiming diskette units, Unit Numbers 0,1 will be declared. This prevents the traditional unit numbering from "shifting" on workstations with 0 or 1 diskette drives installed. There is an "implicit" assumption by OS/2 System Initialization, that OS2DASD will be the first block driver loaded. In addition, if a single diskette drive is installed, OS2DASD creates a pseudo drive B unit, which is mapped to the first diskette drive.

2-24 Logical Drives / Removable Devices

OS2DASD scans the UnitCBs created previously for non-removable drives. The Sector 0 of each drive is read and the partition

record is checked for a file system partition entry. See Boot Record Architecture for more information.

If a file system partition is found, it is considered the *primary partition* on the volume. A VolCB is created and given the next available unit number from this range.

When all non-removable drives have been scanned, sector 0 of each non-removable drive is read again and scanned for an extended volume entry. This entry points to a new extended boot sector on the same drive.

The extended boot sector is read and scanned for a file system partition entry. If an entry is found, then a VolCB is allocated and assigned the next available unit number from this range. The sector is also searched for another extended volume entry.

This process is repeated until the end of the extended volume chain on the drive is reached or no drive letters are left.

When the search is completed, the same search is repeated on the next physical drive.

After all non-removable drives have been processed and the primary or logical drives are allocated, then VolCBs are allocated for any remaining removable drives in the system.

BIOS Parameter Block (BPB) Management

The BPB resides at byte 0 in the first sector of a file system partition. This sector is called the *operating system startup record*. On non-partitioned media, the BPB resides at sector 0, byte 0.

The operating system startup record does not contain a partition table. The partition containing the operating system startup record is pointed to by an entry at the partition table contained in a master or extended boot record.

The BPB is a shared data structure between a block device driver and the FAT file system. In the case of an HPFS, the driver maintains a pseudo BPB.

When the VolCB for the logical drive is created during OS2DASD initialization, the BPB for each logical drive is read.

Validation checks are made by OS2DASD on the BPB to determine if any BPB had been written to the media. In the absence of a valid BPB, OS2DASD creates one, based on the size of the partition. The remainder of the BPB fields are filled in by a table-driven lookup, based on the size of the volume. This BPB is supplied to the OS/2 Kernel and not written to the media.

The driver keeps two copies of the BPB. One representing the BPB determined from the media currently in the drive and one representing the device, assuming its maximum capacity.

For non-removable devices, the BPBs are always identical. For removable devices, the BPBs may differ if the media in the drive is formatted to a lower capacity than the drive is capable of handling.

An example would be a 720KB diskette in a 1.44MB diskette drive. The media BPB is altered to match the media when OS2DASD receives a Build BPB request packet. The device BPB is only altered by using the Category 08h, Function 43 "Set Device Parameters" IOCTL.

This would typically be done by FORMAT which forces a diskette drive to format media at a lower capacity than which it normally operates.

One other idiosyncrasy of BPBs is the *Hidden Sector* field. This is discussed further in the Request Packet Management section.

Request Packet Management

The following section groups request packets by function rather than by numerics.

Removable and Non-Removable Media

04h	Read
08h	Write
09h	Write with Verify
18h	Read (No caching)
19h	Write (No caching)
1Ah	Write with Verify (No caching)

These operations are relatively straightforward, except for the calculation of the location to read.

The disk location to access is calculated as follows:

$$\begin{aligned} \text{Absolute Disk Location} = & \text{Absolute location of Master/Extended Boot Record} \\ & + \text{Hidden Sector Field of Logical Drive BPB} \\ & + \text{RBA Offset in Request Packet} \end{aligned}$$

The *no caching* versions of these commands are handled identically to the regular versions except when commands 18h through 1Ah are processed. To suppress adapter level hardware caching, the appropriate bits are set in the EXECUTE_IO IORB.

Removable Media

01h	Check Media Change
11h	Reset Media Change
0Fh	Check for Removable Media
12h	Get Logical Drive Map
13h	Set Logical Drive Map
01h	Check Media Change
11h	Reset Media Change

01h - Check Media Change, 11h - Reset Media Change: When a Check Media Change packet has been received and the last unit status indicates that a media change has not occurred, then OS2DASD will send a request to the .ADD driver to get an updated status. If the current internal status indicates the media has changed, then "Changed" status will be returned.

OS2DASD retains the result of this call and monitors for a media change indication on other I/O operations.

When a Media Change is detected, the driver retains this information and blocks subsequent I/O requests to the unit until a Reset Media Change packet is received.

0Fh - Check for Removable Media: When OS2DASD receives the Check for Removable Media request, it locates the appropriate VolCB that corresponds to the requested unit number. The VolCB points to the corresponding UnitCB that contains the UnitInfo obtained from the .ADD driver. The BUSY bit of the request packet is set accordingly.

12h - Get Logical Drive Map, 13h - Set Logical Drive Map: OS/2 allows multiple drive letters to be assigned to the same removable device. This assignment occurs automatically for the B drive on a single diskette system and by way of the EXTDSKDD.SYS driver to create additional drive letter aliases for removable drives.

Only one drive letter at a time may be assigned to a removable drive from a set of drive letters that might potentially be mapped to the drive.

The binding of a specific unit number to a removable device is accomplished through the use of Set Logical Drive Map.

When OS2DASD receives the Set Logical Drive Map request, it looks up the physical removable device that should be assigned to this drive letter. It then updates its tables to indicate that the requested unit number owns the removable device.

When OS2DASD receives the Get Logical Drive Map request, it determines the physical drive associated with the unit number provided. OS2DASD then searches the VolCBs to determine which VolCB currently owns the removable device. This owning unit number is returned to the kernel.

The unit numbers returned on both of these packets must be 1-based rather than zero-based. A zero unit number returned indicates that there are no alias drive letters that can be assigned to the removable device.

Non-Removable (Partitionable) Media

16h Get Partitionable Disk Count
17h Map Unit Numbers to Physical Drive

16h - Get Partitionable Disk Count: The number of non-removable drives is returned.

17h - Map Unit Numbers to Physical Drive: The request packet *unit* field indicates a non-removable drive number that is zero-based. OS2DASD adds Function 80h and searches for VolCBs associated with that particular physical unit. A bit map of unit numbers associated with the physical drive is returned. The bit numbering scheme for the bit mask corresponds to $2^{**}(\text{unit number})$ in a *ULONG* field.

Boot Record Architecture

This appendix describes the details of the data that appear on a physical disk. It also describes the structures that are placed on the disk by various utilities.

Master Boot Record

The master boot record is always located on sector 1 of the first track (track 0) on the disk. The following table shows the layout of the various components inside the Master Boot Record. The various components are described below.

Offset	Description	Size
+0	Master Boot Record Program	446 bytes
+446	Partition Table	64 bytes
+510	Signature (55AAH)	2 bytes

Master Boot Record Program

This code is given control from BIOS during boot. Its function is to load the operating system's boot program from the partition that was marked as being startable and turn control over to the (assumed) code that was loaded.

The Master Boot Record Program may be placed on the disk by individual operating systems. If the signature in the Master Boot Record is valid, then the Master Boot Record Program must not be modified. Operating systems must not place requirements on nor make assumptions about the Master Boot Record Program.

Partition Table

This is a vector of 4 structures that allows the disk to be divided up into four distinct areas or partitions. The following table shows how they are arranged in this vector.

Offset	Description	Size
0	Partition 1	16 bytes
16	Partition 2	16 bytes
32	Partition 3	16 bytes
48	Partition 4	16 bytes

It is up to an individual operating system if one of those parts is to be further sub-divided. For example, DOS Version 3.30 implemented a scheme where an "extended partition" could be used to define logical disks to allow the use of larger hardfiles.

The following table shows the format of the individual entries in the partition table. A description of the individual fields follows.

Offset	Description	0	1	2	3
+0	Partion Start	Boot Indicator	Head	Sector	Cylinder
+4	Partion End	System Indicator	Head	Sector	Cylinder
+8	Offset from start of disk (sectors)	Low Word		High Word	
+12	Partion Length (sectors)	Low Word		High Word	

Partition Start

This 4 byte field identifies the beginning of a partition. It also contains an indicator that flags the partition as being active or bootable. This field is composed of several bytes defined as follows.

Boot Indicator

This byte indicates if the partition is active. If the byte contains 00H, then the partition is not active and will not be considered as bootable by the Master Boot Record Boot Program. If the byte contains 80H, then the partition is considered active. The Master Boot Record Boot Program will then attempt to load the first sector described by this partition table entry and transfer control to it. The Master Boot Record Boot Program should only attempt to boot the first partition it finds that is marked active.

Head

This byte contains the number of the first head of the partition.

All partitions are allocated in cylinder multiples and begin on sector 1, head 0.

EXCEPTION: The partition that is allocated at the beginning of the disk should start at cylinder 0, head 1, sector 1, to leave room for the disk's master boot record and other information used to define the fixed disk type on that system. An operating system should not use any data space on cylinder 0 head 0 of a fixed disk.

Sector

This byte contains the sector number of the first sector of the partition. This value should always be 1 (sector numbers are 1 based) for the Partition Begin field because partitions are defined to start on cylinder boundaries. Note that the sector number byte also contains the high order 2 bits of the cylinder number in the high order 2 bits of this byte. Therefore, this byte can have values other than one, but the sector bits of this byte always contains the value 1.

Cylinder

This byte contains the low order 8 bits of the 10 bit cylinder number that indicates the starting cylinder of the partition.

Partition End

This 4 byte field identifies the end of the partition. It also contains an indicator as to which operating system owns the partition. This field is composed of several bytes that are defined as follows.

System Indicator

This byte indicates what operating system owns the particular partition. The values and what they represent are listed in Fixed Disk Partition ID Assignments. A value of 0 indicates an unused entry.

Head

This byte contains the last head number in the last cylinder occupied by this partition.

Sector

This byte contains the sector number of the last sector on the last cylinder occupied by this partition. It also contains the high order two bits of the cylinder number in the high two bits of this byte.

Cylinder

This byte contains the low order 8 bits of the 10-bit cylinder number that indicates the ending cylinder of this partition.

Offset from Start of Disk

This 4-byte field contains the number of sectors preceding each partition on the disk. The value is obtained by counting the sectors beginning with cylinder 0, sector 1, head 0 of the disk, and incrementing the sector, head, and then cylinder values up to the beginning of the partition. Thus, if the disk has 17 sectors per track and 4 heads, and the second partition begins at cylinder 1, sector 1, head 0, the partition's starting relative sector is 68 (decimal)-there were 17 sectors on each of 4 heads on 1 track allocated ahead of it.

The field is stored with the least significant word first.

Partition Length

This 4 byte field contains the number of sectors allocated to the partition. This field is stored least significant word first.

Signature

The last 2 bytes of the boot record (55AAH) are used as a signature to identify a valid boot record containing code that is executable on Intel X86 processors. Both this record and the partition boot records are required to contain the signature at offset 01FEH (510).

Fixed Disk Technical Information

A fixed disk boot record must be written on the first sector of all fixed disks or logical drives within an extended partition and must contain:

- o Code to load and give control to the boot record for one of four possible operating systems.
- o A partition table at the specified offset into the boot record. Each table entry is 16 bytes long, and contains the starting and ending cylinder, sector, and head for each of four possible partitions, as well as the number of sectors preceding the partition and the number of sectors occupied by the partition. The "boot indicator" byte is used by the boot record to determine if one of the partitions contains a loadable operating system. FDISK (or equivalent) initialization utilities mark a user-selected partition as "startable" by placing a value of 80H in the corresponding partition's boot indicator (setting all other partition's indicators to 00H at the same time). The presence of the 80H tells the Master Boot Record Program to load the sector whose location is contained in the following 3 bytes. That sector is the actual boot record for the selected operating system, and it is responsible for the remainder of the system's loading process (as it is from diskette). All boot records are loaded at absolute address 0:7C00.
- o A Signature to indicate a valid Master Boot Record.

System Initialization

The System initialization (or system boot) sequence is as follows:

1. System initialization first attempts to load an operating system from the first diskette drive. If the drive is not ready or a read error occurs, it then attempts to read the fixed disk master boot record from the first sector of the first fixed disk on the system. If unsuccessful, or if no fixed disk is present, it invokes a RIPL device, ROM BASIC or prompts for a startable diskette.
2. If successful, the master boot record is given control. It examines the partition table imbedded within it. If one of the entries indicates a "startable" (active) partition, its boot record is read (from the partition's first sector) and give control.
3. If none of the partitions is startable, a RIPL device or ROM BASIC is invoked or a prompt for a bootable diskette is displayed.
4. If any of the boot indicators are invalid (values other than 00h or 80h) the message **Invalid partition table** is displayed and the system stops. You may then insert a system diskette in drive A and use system reset to restart from diskette.
5. If the partition's boot record cannot be successfully read within five retries due to read errors, the message **Error loading operating system** appears and the system stops.
6. If the partition's boot record does not contain a valid "signature," the message **Missing operating system** appears, and the system stops.

When a partition's boot record is given control, it is passed its partition table entry address in the DS:SI registers.

System programmers designing a utility to initialize/manage a fixed disk must provide the following functions at a minimum:

1. Write the master disk boot record/partition table to the disk's first sector to initialize it if it is not already present.
2. Perform partitioning of the disk—that is, create or update partition table information (all fields for the partition) when the user wishes to create, modify, or remove a partition. This may be limited to creating a partition for only one type of operating system, but must allow repartitioning the entire disk, or adding a partition without interfering with existing partitions (user's choice).

Note: When changing the size or location of any partition, you must ensure that all existing data on that partition has been backed up (the partitioning process will "lose track" of the previous partition

boundaries).

3. Provide a means for marking a user-specified partition as startable, and resetting the startable indicator bytes for other partitions at the same time.
4. Such utilities should not change or move any partition information that belongs to another operating system.

Fixed Disk Partition ID Assignments

Partition	Description
00	Unused Partition
01	DOS, 12-bit FAT
02	XENIX System, includes SCO/XENIX
03	XENIX User, includes SCO/XENIX
04	DOS, 16-bit FAT
05	DOS and OS/2, >32MB support; defines an Extended partition which may include other partition types.
06	DOS, >32MB support, up to 64K Allocation unit
07	OS/2, >32MB partition support (IFS)
08	AIX
08	OS/2 (thru Version 1.3 only)
08	DELL partition spanning multiple drives (array)
08	Commodore DOS Partition
09	AIX
0A	OS/2 Boot Manager Partition
0B - 0D	Available for assignment
0E - 0F	Reserved
10	Reserved
11	OS/2 Boot Manager: DOS - Inactive type 1

12	Reserved
13	Available for assignment
14	OS/2 Boot Manager: DOS - Inactive type 4
15	Available for assignment
16	OS/2 Boot Manager: DOS - Inactive type 6
17	OS/2 Boot Manager: DOS - Inactive type 7
18 - 20	Available for assignment
21	Reserved
22	Available for assignment
23 - 24	Reserved
25	Available for assignment
26	Reserved
27 - 30	Available for assignment
31	Reserved
32	Available for assignment
33 - 34	Reserved
35	Available for assignment
36	Reserved
37 - 3F	Available for assignment
40	Series/1 Disk
41	Personal RISC Boot Partition

42 - 4F	Available for assignment
50	OnTrack Disk Manager
51	OnTrack Disk Manager
52	Reserved
53 - 55	Available for assignment
56	Reserved
57 - 60	Available for assignment
61	Reserved
62	Available for assignment
63	SCO UNIX
64	Novell
64	Speedstore
65	Novell 286 Netware
66	Novell 386 Netware
67	Novell (future use)
68	Novell (future use)
69	Novell (future use)
6A - 70	Available for assignment
71	Reserved
72	Available for assignment
73 - 74	Reserved
75	PC/IX

76	Reserved
77 - 79	Available for assignment
80 - 81	Reserved
82	Prime
83	Apple Computer
84	System Hibernation for APM 1.1
85 - 85	Available for assignment
86	Reserved
87	HPFS FT mirrored partition
88 - 92	Available for assignment
93 - 94	Reserved
95 - A0	Available for assignment
A1	Reserved
A2	Available for assignment
A3 - A4	Reserved
A5	Available for assignment
A6	Reserved
A7 - B0	Available for assignment
B1	Reserved
B2	Available for assignment
B3 - B4	Reserved
B5	Available for assignment

B6 - B8	Reserved
B9 - C0	Available for assignment
C1	Reserved
C2 - C3	Available for assignment
C4	Reserved
C5	Available for assignment
C6	Reserved
C7	HPFS FT disabled mirrored partition
C8 - D7	Available for assignment
D8	CP/M 86
D9 - DA	Available for assignment
DB	Reserved
DC - E0	Available for assignment
E1	Speedstore
E2	Available for assignment
E3	Storage Dimensions (Maxtor Retail Subsidiary)
E4	Speedstore
E5 - E6	Reserved
E7 - F0	Available for assignment
F1	Storage Dimensions (Maxtor Retail subsidiary)
F2 - F3	Reserved
F4	Storage Dimensions (Maxtor Retail subsidiary)

F5	Available for assignment
F6	Reserved
F7 - FD	Available for assignment
FE	IBM PS/2 IML
FF	Bad Block Tables - Must be on cylinder 0

Extended DOS Partition

Fixed disks can be divided into primary partitions, and an extended partition that contains multiple logical block devices. The *extended partition* is indicated by a System ID byte of 05h in the partition table of the Master Boot Record. This partition cannot be started, and programs that can set startable partitions (such as OS/2 FDISK) do not allow the partition to be marked as able to start.

The extended DOS partition can be created only if a primary DOS partition already exists on a startable drive. A primary partition is a partition with a System ID byte of 01h, 04h, 06h, or 07h. If the drive cannot be started, then an extended DOS partition can be created without having a primary DOS partition.

Note:

1. FDISK refers to extended volumes as logical drives.
2. This extended partition support can be used on any fixed disk supported by the OS/2 operating system.

The extended DOS partition starts and ends on a cylinder boundary, and contains a collection of extended volumes that are linked together by a pointer in the extended volumes' extended boot record. An *extended volume* consists of an extended boot record and one logical block device. In OS/2 Version 1.0, an extended volume could not be larger than 32MB, due to the limitations of the FAT file system. However, in OS/2 2.0 and 2.1, this restriction has been removed. An extended volume created within the extended DOS partition can be any size, from one cylinder long through the maximum available contiguous space in the extended DOS partition. All extended volumes must start and end on a cylinder boundary. The extended boot record corresponds to the Master Boot Record at the beginning of an actual physical disk. The logical block device corresponds to the DOS partition that is pointed to by the Master Boot Record.

The logical block device begins with a normal DOS boot sector if it is a DOS logical block device (System ID=1, 4, or 6). Installable File System (IFS) logical block devices (System ID=7) need not start with a normal DOS boot sector. This logical block device must start on a cylinder and head boundary and must follow the extended boot record on the physical disk. The logical block device and the extended volume both end on the same cylinder boundary.

Each extended volume contains an extended boot record located in the first sector of the disk location assigned to it. This extended boot record contains the 55AAh signature ID byte. This allows programs that look at the Extended (Master) Boot Record to be compatible. This extended boot record also contains a partition table, which can contain only two types of entries. The boot code is not critical, as the devices are not considered startable. The boot code can simply report a message indicating an unstartable partition if it is executed.

The partition table portion of the extended boot record is the same as the

partition table structure in the Master Boot Record. This structure has four partition entries of 16 bytes each. The System ID byte must be filled in for all four entries with one of the following values:

00h No space allocated in this entry.

01h DOS partition up to 16MB.

04h DOS partition with 32MB > SIZE > 16MB.

05h Maps out area assigned to the next extended volume. Serves as a pointer to the next extended boot record.

06h DOS partition size > 32MB.

07h Installable file system.

If the System ID byte is 0, then the values in that partition table entry are set to 0.

If the operating system detects any values other than 01h, 04h, 06h, or 07h, it ignores that entry and does not attempt to install the logical block device. This allows future expansion of devices in this area without problems of compatibility with earlier systems.

The partition start and end fields Cylinder, Head, and Sector (C,H,S) are filled in for any of the four partition entries in an extended boot record that have one of the System ID bytes. This allows a program such as FDISK to determine the allocated space in the extended DOS partition, and allows the physical device drivers to determine the physical DASD area that belongs to it. The partition start and end fields (C,H,S) for the partition entry that points to the logical block device (System ID 01h, 04h, 06h, or 07h) map out the physical boundaries of the logical block device. They are offset relative to the beginning of the extended boot record that the entry resides in. The partition start and end fields for the partition entry that points to the next extended volume (System ID 05h) map out the physical boundaries of the next extended volume. They are relative to the beginning of the entire physical disk.

The *relative sector* and *number of sector* fields are set up differently depending on what System ID byte is used. If 01h, 04h, 06h, or 07h is in the *System ID* field for that extended partition entry (pointer to the logical block device), the *relative sector* field is set up as an offset from (and including) the start of the extended boot record for the associated extended volume. The *number of sectors* field is filled in with the size of the created logical block device area (that is, the number of sectors mapped out by the start and stop cylinder/track/sector fields). The size of the extended volume can be calculated by adding the *relative sector* field and the *sector size* field of the associated extended boot record.

If the System ID byte is 05h, then the *relative sector* field is the offset (of the next extended volume) in sectors from the start of the entire extended DOS partition. The *number of sectors* field is not used in this field, and is filled with 00hs.

This architecture allows only one logical block device to be defined for each extended boot record. Therefore, a maximum of two partition entries at a time is used in each extended boot record - an entry with System ID byte of 01h, 04h, 06h, or 07h, and an entry with ID of 05h (which is the pointer to the next extended volume).

Although only two entries can be used, a program installing these devices does not assume that the first two entries will be the non-zero entries.

Installing Block Devices in the Extended DOS Partition

To install block devices, physical device drivers first install the primary DOS partitions on all physical drives, if any exist. This ensures that an existing drive letter, **D:**, on the 81h drive remains the same. After these devices are installed on the 80h drive, the drivers look for the existence of the extended DOS partition. If one exists, then the physical device drivers look at the first sector of the extended DOS partition for the first extended boot record. If there is a valid System ID (01h, 04h, 06h, or 07h) in any of the four partition entries, the device is installed and assigned the next available drive letter. This occurs before any CONFIG.SYS device drivers are loaded, so the FDISK will correctly display the drive letter when space is allocated for the drive.

The first extended boot record (in the extended DOS partition) is a special case, because it is possible there will not be a device to be installed defined in the partition table. The first device might have been created and then deleted at some time. However, the first extended boot record is needed to point to the next one, if one exists. Any other extended boot record will always have a device to be installed.

Once a device has been installed (or the special cases above occur), the physical device driver searches the other partition entries for a System ID byte of 05h, indicating that another device (extended volume) exists. If a 05h is not found, there are no more logical block devices (extended volumes) in the extended DOS partition.

If a 05h System ID is found, the start location in that partition entry is read in order to find the location of the next extended boot record. When located, it is read in, and then the process is repeated in order to install additional devices.

Once all the valid devices for a physical drive have been installed, the next physical drive is examined and the entire process is repeated.

A device driver does not assume any order dependency when searching for a particular System ID byte in an extended boot record. All four possible entries in an extended boot record partition table are searched, before a driver decides that a particular System ID byte does not exist.

The extended DOS partition can only be created if a primary DOS or IFS partition already exists on a bootable drive. A *primary DOS partition* has a System ID of 01h, 04h, or 06h. A *primary IFS partition* has a System ID of 07h. If the drive is not bootable, an extended DOS partition can be created without having a primary DOS partition. The extended DOS partition starts and ends on a cylinder boundary.

Creating Block Devices in the Extended DOS Partition

To create the structure for an extended volume in the extended DOS partition, FDISK determines if there is available space in the extended DOS partition and if less than 24 total devices are allocated in the system. The maximum number of block devices allowed is 26, and two are used by diskettes, **A:** and **B:**. The program then creates an extended boot record at the space located, with a partition entry filled in (with the size and location information) for that logical block device. If this is not the first extended boot record, the program backs up to the last extended boot record in the chain (as linked by the 05h entries), and creates a partition entry in that extended boot record that has the size and location data for the newly created record. This action creates the pointer required to locate the newly created boot record.

If this is the first extended boot record in the extended DOS partition only the size, type, and location of the logical block device needs to be put into a partition entry. The start of the extended DOS partition in the Master Boot Record serves as a pointer to this extended volume.

Deleting Block Devices in the Extended DOS Partition

To delete a block device, the program sets the 16-byte partition entry that contained the System ID byte, to 0. If in the same extended boot record there exists a partition entry with System ID of 05h, indicating that another extended volume exists, this information is copied to the 05h partition entry of the previous extended boot record. (See the following figure for further information.)

Note: There is one exception to this rule. If the deleted logical block device is at the beginning of the extended DOS partition, only the partition entry indicating the device type is set to 0. The 05h pointer information is to be left in place.

```

Master Boot Record.....Note 1....
.....
.....Note 2    4    2    5    0    55AA    Note 3

Primary DOS Partition Note 4
DOS C: drive  32MB    Size

Other Operating System Partition
(XENIX)      Note 5

E
x
t
D
O
S
P
a
r
t
i
t
i
o
n
Extended Boot Record..Note 6.....
.....
.....Note 7    4    5    0    0    55AA
LOGICAL Block Device D:      Note 8
32MB    Size  16MB or IFS
Extended Boot Record..Note 9.....
.....
.....Note 10   1    5    0    0    55AA
LOGICAL Block Device E:
Size    16MB
Extended Boot Record.....
.....
```

.....Note 11 6 5 0 0 55AA

Area reserved for future CP/DOS use
Note 12

Extended Boot Record.....

.....

.....Note 13 4 0 0 0 55AA

LOGICAL Block Device G:
32MB Size 16MB

Free Space in Extended Partition

Free Space not allocated to any
partition

- Note 1** Master Boot Record code, starting at Track 000, Head 00, Sector 01 of disk 80h or 81h.
- Note 2** Partition table for Master Boot Record. See BPB and Get Device Parameters for Extended Volumes for the layout. The 4 is the System ID byte in the partition table that indicates a DOS partition greater than 16MB and less than or equal to 32MB. The 2 is a XENIX** partition, and the 05h maps the extended DOS partition.
- Note 3** 55AAh is the signature to validate the Master Boot Record.
- Note 4** Primary DOS area, which must reside entirely in first 32MB of the disk. **C:** is block device 80h. **D:** is block device 81h, if it exists. This partition has a maximum size of 32MB.
- Note 5** Other operating system on disk.
- Note 6** Extended boot record for extended volume that corresponds to logical block device **D:**. (This assumes only the 80h block device exists.) If the 81h block device exists, this would be block device **E:**.
- Note 7** Logical block device **D:** partition table entry. This has a maximum size of 32MB, which is indicated by the System ID of 4. This must set the logical DOS block device as starting at the next track boundary. The 05h System ID byte in the second partition entry maps out the space allocated to the next extended volume. The starting cylinder/sector/head in the partition entry with an ID of 05h is the location of the next extended boot record of the next extended volume.
- Note 8** Logical block device **D:**. Logical DOS devices and the primary DOS

partition always begin with a DOS boot record.

Note 9 Extended boot record for logical block device **E:**.

Note 10 Partition table entry for logical block device **E:**. This logical DOS block device is less than or equal to 16MB, as indicated by the System ID of 01h. The entry with System ID of 05h maps out the space allocated to the next extended volume.

Note 11 The System ID byte of 06h indicates a logical block device greater than 32MB. This block device is indicated by a block device letter of **F**. Note also that a pointer to the next extended volume exists.

Note 12 The greater than 32MB FAT partition.

Note 13 Partition table entry for final DOS logical block device. Note that the absence of the 05h ID byte means that there are no other extended volumes allocated in the extended DOS partition. This would have a block device letter of **G:**, if the previous logical block device was recognized. Otherwise it would be **F:**.

Offs	Purpose		Head	Sector	Cylinder
1BE	Partition 1 begin	boot ind	H	S	CYL
1C2	Partition 1 end	syst ind	H	S	CYL
1C6	Partition 1 rel sect	Low word		High word	
1CA	Partition 1 # sects	Low word		High word	
1CE	Partition 2 begin	boot ind	H	S	CYL
1D2	Partition 2 end	syst ind	H	S	CYL
1D6	Partition 2 rel sect	Low word		High word	
1DA	Partition 2 # sects	Low word		High word	
1DE	Partition 3 begin	boot ind	H	S	CYL
1E2	Partition 3 end	syst ind	H	S	CYL
1E6	Partition 3 rel sect	Low word		High word	
1EA	Partition 3 # sects	Low word		High word	

1EE Partition 4 begin	boot ind	H	S	CYL
1F2 Partition 4 end	syst ind	H	S	CYL
1F6 Partition 4 rel sect	Low word			High word
1FA Partition 4 # sects	Low word			High word
1FE Signature				

BPB and Get Device Parameters for Extended Volumes

For purposes of the BIOS Parameter Block (BPB) and Get Device Parameters (generic IOCTL), an extended volume appears to the system as a fixed disk. The extended boot record corresponds to the Master Boot Record of a real fixed disk and the logical block device corresponds to the primary DOS partition.

This means the BPB of the logical DOS block device of the extended volume describes the environment in the extended volume. This consists of the extended boot record and the logical block device. The meaning of the fields is consistent with the meaning of the fields for the primary DOS partition; they relate to the entire physical disk, the primary DOS partition, and the Master Boot Record. For example, the number of hidden sectors is the distance from the beginning of the extended boot record (of the extended volume in question) to the start of the logical DOS block device (the DOS Boot Record). The number of sectors field describes only the logical block device, just as it normally only describes the primary DOS partition.

Category 08h Generic IOCtl Commands

The philosophy described above also applies to the disk generic IOCtl commands. For any logical block device of an associated extended volume, physical cylinder, head, and sector I/O is mapped to within the extended volume - Cylinder 0, Head 0, Sector 1 is mapped to the extended boot record. An error condition is generated for any attempt to do C,H,S I/O beyond the size of the extended volume in question.

Category 09h Generic IOCtl Commands

Category 09h generic IOCtl commands are used to access the entire physical fixed disk without consideration of logical volumes. Physical cylinder, head, and sector begin at the start of the physical drive, instead of at the beginning of an extended volume.

Type 6 Partition

A 12-bit or 16-bit type FAT can be used to map a Type 6 partition because the type of FAT is based strictly on the number of allocation units (clusters), and is the same algorithm used to define the type of FAT in the OS/2 Version 1.0 operating system. FAT cluster sizes are based on powers of 2. Assuming usage of the OS/2 FORMAT utility, the minimum cluster size for a hard file is 2KB. Cluster size and the type of FAT (12-bit versus 16-bit) are determined by the media partition size. The OS/2 FORMAT algorithm is:

```
If partition size <= 16MB
then;
    use 12-bit FAT;                /* max 4084 entries */
    max cluster size = 4KB;
end;
else;                             /* partition size >16MB */
    use 16-bit FAT;               /* max 64KB entries */
    min cluster size = 2KB;
end;
```

The actual determination of the partition type is made based on the number of clusters on that partition. OS/2 FORMAT makes sure that this is true for the <16MB and >16MB partitions.

```
If number of clusters <= 4084
    use 12-bit FAT;                /* max 4084 entries */
else
    use 16-bit FAT;               /* max 64KB entries */
```

A partition size of 128MB requires a 2KB cluster size, based on a maximum of 64KB allocation units (clusters). A partition size in the range of 129MB and 256MB requires a 4KB cluster size, based on 64KB allocation units. A partition size in

the range of 257MB and 512MB requires an 8KB cluster size, based on 64KB allocation units.

The configuration table used by OS/2 FORMAT is show in the following table:

Total # of Sectors	Size of Partition	Sector Cluster	# of Root DIR Entries
32K	16MB	8	512
64K	32MB	4	512
256K	128MB	4	512
512K	256MB	8	512
1M	512MB	16	512
2M	1GB	32	512
4M	2GB	64	512
8M	4GB	128	512

Note: For Type 6 partitions, it is safe to use a non-default configuration, but this might be unsafe for other partition types.

The partition can reside anywhere on the media, as the primary DOS partition, or as an extended volume within the extended DOS partition. The BPB parameter *number of sectors per FAT* field width has been extended from a byte to a WORD, in order to define a full 128KB FAT structure. This change affects all DOS partition types.

Layout of Block Devices with a Type 6 Partition Using XENIX

```
Master Startup Record...Note 1....
.....
.....Note 2    2  6  0  0  55AA  Note 3

Other Operating System Partition
(XENIX)
Size    32MB  Note 4

Primary DOS Partition Note 5
DOS C: drive  32MB    Size

Free Space
(not allocated to any partition)
```

Note 1 Master Startup (Boot) Record code, starting at Track 000, Head 00, Sector 01 of disk 80h or 81h.

Note 2 Partition table for Master Startup (Boot) Record. The 2 is the System ID byte in the partition table that indicates a XENIX partition, and the 06h map indicates a primary DOS Type 6 partition.

Note 3 55AAH is the signature to validate the Master Startup (Boot) Record.

Note 4 Other operating system (XENIX) on disk.

Note 5 Primary DOS partition. **C:** is block device 80h. The partition type in this example is a 6, because it ends beyond the first 32MB of the disk. Within the scope of this definition, though the size of a primary DOS partition can be less than 32MB (because it ends beyond the first 32MB of the disk), it is defined as a Type 6.

Layout of Block Devices with a Type 6 Partition

```
Master Startup Record...Note 1....
.....
.....Note 2    6  0  0  0  55AA    Note 3

Primary DOS Partition Note 4
DOS C: drive  Size   32MB
```

- Note 1** Master Startup (Boot) Record code, starting at Track 000, Head 00, Sector 01 of disk 80h or 81h.
- Note 2** Partition table for Master Boot Record. The 6 is the System ID byte in the partition table that indicates a DOS partition where SIZE > 32MB.
- Note 3** 55AAh is the signature to validate the Master Startup (Boot) Record.
- Note 4** Primary DOS area. Owns the entire media and exceeds 32MB in size. **C:** is block device 80h.

Type 7 Partition

Partition Type 7 is used for Installable File Systems only. The internal FAT file system should not use this partition type because older versions of the DOS and OS/2 operating systems will not be able to access the partition.

Extended Device Driver Interface Specification

The Extended Device Driver interface supported in OS/2 2.1 is specifically targeted for service of hard disk devices. This interface can:

- o Support submission of multiple asynchronous requests
- o Allow physically discontinuous data transfer areas
- o Support a new class of disk devices with advanced bus-mastering and intelligent transfer capabilities
- o Allow physical device drivers to optimally service large numbers of requests in a heavily loaded environment
- o Provide a mechanism and supporting semantics to support prioritization of request services

The Extended Device Driver interface employs a Request List of prioritized commands the driver can reorder to optimize disk access, subject to prioritization requirements. The requests can also be grouped by the kernel for notification callout from the device driver. In addition, READ and WRITE operations use scatter/gather descriptors, which allow for data transfer to and from discontinuous data buffers. The interface is used asynchronously, removing the need for blocking in the physical device driver or the physical device driver manager when the I/O request itself is asynchronous.

While the asynchronous, multi-request aspects of the interface contribute greatly to overall system performance on existing hardware, scatter and gather is specifically targeted for new classes of disk device hardware. This new class of devices supports transfer from disk to physically discontinuous memory space much more efficiently than alternative implementations.

The importance of this relative efficiency is amplified by the introduction of paging to the OS/2 operating system, where linearly contiguous memory normally maps to lists of discontinuous physical addresses. In addition, this interface is designed and optimized for server environments such as LAN Server 2.0, where file service paths are Ring 0 only and can execute at task or interrupt time. In such an environment, it must be possible to queue requests to the disk driver for service in the context of a network interrupt.

Extended physical disk device drivers refer to a superset of the standard OS/2 1.x physical disk device drivers. The term *standard request* is used to refer to the old style request packets format. The term *extended request* is used to refer to the new request packet format.

Disk Device Driver Architecture

Driver architecture centers around the need to provide fast, efficient services to a file system in a paged environment. In addition, consideration for the needs of a network file server has influenced aspects of the design; however, the requirements are identical in a local-only or workstation environment (that is, a direct, asynchronous, zero-overhead interface between the file system and the supporting physical disk device drivers).

In the OS/2 operating system, a physical disk device driver receives requests for service through a strategy routine at task time in the context of the requestor (an OS/2 thread). The thread of control is also obtained, in the context of interrupts generated by the disk controller, at the physical device driver interrupt routine. In the extended architecture, a second strategy routine is introduced, which can be called directly by File System Drivers (FSDs) at task time or in the context of an arbitrary interrupt. This yields a set of new requirements.

Standard OS/2 Strategy Routine

The standard OS/2 strategy routine is essentially unchanged in this architecture. Underlying queueing mechanisms used by the driver in the strategy routine are modified to support an environment where there are normally two kinds of requests in the queue, standard and extended.

Extended Strategy Routine

The extended strategy routine entry point can be called at interrupt or task time. At interrupt time, the driver can work only with physical addresses or global virtual addresses; therefore, only physical and global virtual addresses, which reference structures that are physically contiguous in memory, are used in this interface. Physical addresses are used for data transfer areas. Global virtual addresses are used for request packets, control structures, and entry points.

Much of the performance gain realized in this interface is dependent on the asynchronous processing of requests. The performance gain realized is degraded considerably if the driver blocks in the strategy routine, forcing the request to be synchronous. Therefore, while it is not required that the driver never block, it is very strongly recommended if performance is of interest in the system for which the driver is targeted. Additionally, if the driver has set the *does-not-block* bit in the DD_DriverCaps field returned from the GET DRIVER CAPABILITIES command, then the driver absolutely must not block in any code path accessible by the extended strategy routine entry point. Furthermore, it must not call any DevHlp routine that could block, effectively limiting the DevHlps that can be called to only those permissible at interrupt time. Among the DevHlps that could block are Lock and Unlock, so all buffers passed through the extended entry points are guaranteed to be locked.

The extended strategy routine is used only to pass requests in Request List form (see Request Lists and Request Control). The physical device driver queues the requests passed, services the controller as necessary, sets status fields in the requests, and returns.

Sorting and Priority

Requests should be sorted into internal queues based on physical disk location and I/O priority to optimize request servicing. Lower priority requests should be satisfied only where they do not significantly slow service of higher priority requests (for example, when a lower priority request refers to a sector in the same cylinder as a high priority request). Because the format of requests in Request Lists is different from the format of standard OS/2 requests, the queue management DevHlp routines cannot be used.

The queueing strategy adopted by the driver is highly dependent on the devices it services. In general, efficiency and request priority are the primary concerns in determining a queueing strategy. Lower priority requests should never slow service of higher priority requests. Lower priority requests can be serviced in the context of servicing higher priority requests, so long as no time-costly operations are necessary to service the lower priority requests. In addition, where contention for resources such as auxiliary, driver-allocated buffers or space on a controller buffer is an issue, higher priority requests should be given preference.

Request Management

The physical device driver needs to manage both requests submitted to the extended strategy routines with the Request List format and requests submitted to the standard strategy routine with the standard request packet format. Notice that the CommandCode field in the OS/2 standard request packet coincides with the CommandPrefix byte in the extended request packet, which is guaranteed to be the ExecuteChain prefix. This allows the driver to manage both requests on the same queue.

Removable Media

The physical device driver should support requests targeted for removable media in the same way it supports requests for nonremovable media.

Devices Not Capable of Scatter/Gather

Although this interface is clearly targeted for a disk device controller capable of scatter/gather, even devices that are not capable of scatter/gather still realize overall system performance gains as a result of supporting multi-request, asynchronous I/O, and interrupt-time execution. The driver is responsible for deciding how to emulate scatter/gather. In general, emulating scatter/gather to programmed I/O devices introduces no significant overhead.

However, if DMA devices, which do not support scatter/gather, attempt to emulate scatter/gather by mapping each scatter/gather descriptor in a single request to one DMA operation, performance will be poor. The driver is better off staging transfers through a pair of contiguous buffers. One buffer can be serviced by the controller while the other is being set up for subsequent operations. While there is overhead incurred in block-copying data through the staging buffer, this is no worse than the overhead the file system would incur in contiguous cache blocks before submission to a standard OS/2 device driver.

Identifying Extended Device Drivers and Capabilities

If the physical device driver is an extended device driver, it recognizes the GET DRIVER CAPABILITIES command. This command returns a characteristics bit field, which describes functional capabilities of the device driver, and an entry point vector, which includes the extended strategy routine and several device driver control routines. If the device driver does not recognize the command or does not respond appropriately, the kernel and all client FSD assume the physical device driver supports only standard OS/2 requests and restricts its behavior appropriately.

GET DRIVER CAPABILITIES Command

This command is structured as a standard OS/2 request packet. Note that the fields prefixed by *DD_* are filled in by the physical device driver. (See the following table.)

Field	Length
Request Header	13 BYTES
Reserved. Must be zero.	3 BYTES
DD_CapStruct	DWORD
DD_VolCharStruct	DWORD

DD_CapStruct	A 16:16 virtual pointer to the Driver Capabilities Structure. This pointer is filled in by the physical device driver. See Driver Capabilities Structure (DCS) for the format of this structure.
DD_VolCharStruct	A 16:16 virtual pointer to the Volume Characteristics Structure (VCS) for this volume. This pointer is filled in by the physical device driver. See Volume Characteristics Structure (VCS) for the format of this structure.

Driver Capabilities Structure (DCS)

The Driver Capabilities Structure (DCS) is maintained by the physical device driver and is passed by reference to the kernel and client FSDs in the GET DRIVER CAPABILITIES command. The kernel and client FSDs must not modify the structure, as it is shared by FSDs and the physical device driver. A DCS has the following format:

Field	Length
Reserved. Must be zero.	WORD
DD_VerMajor	BYTE
DD_VerMinor	BYTE
DD_Capabilities	DWORD
DD_Strategy2	DWORD
DD_SetFSDInfo	DWORD
DD_ChgPriority	DWORD
DD_SetRestPos	DWORD
DD_GetBoundary	DWORD

DD_VerMajor	The major version number of the interface the physical device driver supports, equal to 01h in the first release. Old major versions do not function correctly with a file system using a newer version.				
DD_VerMinor	The minor version number of the interface the physical device driver supports, equal to 01h in the first release. Old minor versions support a strict subset of the functionality found in newer versions.				
DD_Capabilities	A bit field describing the capabilities of the physical device driver: <table><tr><td>Bits 0-2</td><td>Reserved. Must be zero.</td></tr><tr><td>Bit 3</td><td>If set, supports disk mirroring.</td></tr></table>	Bits 0-2	Reserved. Must be zero.	Bit 3	If set, supports disk mirroring.
Bits 0-2	Reserved. Must be zero.				
Bit 3	If set, supports disk mirroring.				

Bit 4	If set, supports disk duplexing.
Bit 5	If set, driver does not block in Strategy2. LAN Server and LAN Manager products using the HPFS 386 file system require that bit 5 be set to guarantee that the physical device driver does not block in Strategy2.
Bits 6-31	Reserved. Must be zero.

DD_Strategy2	The 16:16 entry point for the strategy routine that supports multi-request asynchronous I/O.
DD_SetFSDInfo	The 16:16 entry point for DD_SetFSDInfo. The value returned is 0:0, if the service is not provided by the physical device driver.
DD_ChgPriority	The 16:16 entry point for DD_ChgPriority. The value returned is 0:0, if the service is not provided by the physical device driver.
DD_SetRestPos	The 16:16 entry point for DD_SetRestPos. The value returned is 0:0, if the service is not provided by the physical device driver.
DD_GetBoundary	The 16:16 entry point for DD_GetBoundary. The value returned is 0:0, if the service is not provided by the physical device driver.

Volume Characteristics Structure (VCS)

The parameters passed in the Volume Characteristics Structure (VCS) are used by FSDs to optimize disk access and placement of file system structures on an advisory basis. All values reflect the physical parameters of the logical volume, as if it were a single physical device (that is, whether the media is partitioned or not). This data structure is passed by reference and is maintained and updated by the physical device driver, as necessary. It is expected that the physical device driver would maintain a separate VCS for each logical volume supported. A VCS has the following format:

Field	Length
VolDescriptor	WORD
AvgSeekTime	WORD
AvgLatency	WORD
TrackMinBlocks	WORD
TrackMaxBlocks	WORD
Head Per Cylinder	WORD
VolCylinderCount	DWORD
VolMedianBlock	DWORD
MaxSGList	WORD

VolDescriptor A bit field, defined as follows:

Bit 0	If set, volume resides on removable media
Bit 1	If set, volume is read-only
Bit 2	If set, average seek time independent of position (RAM disk)
Bit 3	If set, outboard cache supported
Bit 4	If set, scatter/gather supported by adapter
Bit 5	If set, ReadPrefetch supported
Bits 6-15	Reserved, set to 0

AvgSeekTime The average seek time (in milliseconds) in servicing this volume. If the seek time is unknown, FFFFH is to be

specified. Can be 0 for RAM disks.

AvgLatency	The average rotational latency (in milliseconds) for the device servicing this volume. If the average latency is unknown, FFFFH is to be specified. Latency can be 0 for RAM disks.
TrackMinBlocks	The number of blocks available on the smallest capacity track; if unknown or not applicable, a value of 1 is specified.
TrackMaxBlocks	The number of blocks available on the largest capacity track; if unknown or not applicable, a value of 1 is specified.
Heads Per Cylinder	The number of heads per cylinder; if unknown or not applicable, a value of 1 is specified.
VolCylinderCount	The number of cylinders in the volume; if unknown or not applicable, the number of allocation blocks (sectors) is used.
VolMedianBlock	The number of the block, which is in the center of the volume with respect to seek time (that is, the block with the smallest average seek time).
MaxSGList	The maximum number of scatter and gather list entries, which can be directly submitted to the adapter servicing this volume with one low-level I/O command. File systems submitting extended commands with scatter and gather lists greater than MaxSGList entries must ensure that the cumulative byte count of each MaxSGList entry in the list is a multiple of the sector size. This field is set to 0, if the volume is serviced by an adapter that does not directly support scatter/gather lists. See Scatter/Gather Descriptor for details on scatter/gather lists passed in extended requests.

Request Lists and Request Control

In order to support multi-request asynchronous I/O, a new request format has been defined, called Request Lists. This format allows multiple requests to be submitted in one call to the extended strategy routine, as well as grouping of those requests for notification purposes. In general, requests from any and all lists can be reordered and considered independently by the physical device driver for optimal throughput. Presently, only READ, WRITE, and READ PREFETCH commands have been defined in the new format.

Through Request Control flags, optional restrictions can be set on the requests to force sequential execution and to allow early termination of request processing, should any of the requests fail.

The notification mechanism allows the kernel and client file systems to receive callout notification when specific individual requests complete, when the entire request list completes, or both. In addition, notification can take place when an error condition occurs, when requests complete successfully, or both. Alternatively, no callout notification can be specified, allowing the system to poll for request completion during idle time.

Extended disk driver requests are submitted directly through the extended strategy routine entry point, DD_Strategy2, obtained through the GET DRIVER CAPABILITIES command and passed to client FSDs through FS_MOUNT. Requests are submitted in request list format, with ES:BX containing a global pointer to the Request List.

Each request in the list is an EXECUTE CHAIN command containing the EXECUTE CHAIN command prefix at the same offset into the extended request packet as the Command field in the standard request packet.

Request Lists have the following format:

Field	Length
Request List Header	20 BYTES
Requests	ARRAY

The Request List header has the following format:

Field	Length
ReqListCount	WORD

Reserved	WORD
LstNotifyAddress	DWORD
LstRequestControl	WORD
Block Device Unit	BYTE
LstStatus	BYTE
DDReserved	DWORD
DDReserved	DWORD

ReqListCount The number of requests in the Request List.

Reserved Must be 0.

LstNotifyAddress The 16:16 address of a notification routine to be called (according to the flags in LstRequestControl) when all requests have been completed or terminated due to error conditions. LstNotifyAddress is not valid if bits 4 and 5 of LstRequestControl are clear. LstNotifyAddress is called with the following parameters:

ES:BX 16:16 Address of Request List header

CF Set, if an unrecoverable error has occurred

The physical device driver is responsible for saving and restoring any registers that must survive the call.

LstRequestControl A bit field of control flags as follows:

Bit 0 Reserved.

Bit 1 If set, there is only one request in the list. The same mechanism is used to submit one or many requests.

Bit 2 If set, requests are to be executed in sequence. Indicates that requests in this list must be executed in the order in which they appear in the Request List. They need not be executed adjacently; requests from other lists can interleave execution of this list.

Bit 3 If set, terminate on error. Indicates that, if an unrecoverable error occurs in processing any request in the list, outstanding

requests in the list must not be executed. All Status, ErrorCode, and BlocksXferred fields must be updated, however.

Bit 4 If set, notify immediately on error only. Indicates that the request list notification routine, LstNotifyAddress, should be called immediately if an unrecoverable error occurs in servicing any of the requests in the Request List.

Bit 5 If set, notify on completion. Indicates that the request list notification routine should be called when execution of the requests has completed, regardless of error conditions.

If bit 4 and bit 5 are clear, the request list notification routine address is not valid. If bit 4 or bit 5 is set, the notification routine address is valid.

Bits 6-15 Reserved. Must be 0.

Block Device Unit The logical unit number of the volume the requests are directed to. Note this forces all requests in the list to be addressed to the same block device unit.

LstStatus Indicates the overall status for the Request List. This field should be set immediately by the physical device driver at strategy time and updated as requests complete successfully or unsuccessfully.

The low nibble indicates the completion status of the requests in the list, giving the LstStatus byte the following values:

X0h Indicates that no requests have been queued. It is guaranteed that the kernel will set this status on entry to the physical device driver. The physical device driver sets this status on return only if queueing was not possible due to exhausted driver-internal resources.

X1h Indicates that some requests have not yet been sorted into internal queues. That is, queueing is in process but has not yet been completed.

X2h Indicates that all requests in the list have been queued and are awaiting service.

X4h Indicates that all requests in the list have been completed successfully or unsuccessfully due to error conditions.

X8h Reserved.

The high nibble indicates the error status of the requests in the list, giving the LstStatus byte the following values:

0Xh No error.

1Xh Indicates that an error has occurred in processing at least one of the

requests, but the physical device driver has successfully recovered the error through retries, ECC, disk mirroring, or duplexing.

2Xh Indicates that an unrecoverable error has occurred in processing at least one of the requests.

3Xh An unrecoverable error has occurred with retry.

4Xh Reserved.

8Xh Reserved.

Bits in the high nibble can be set in combination with bits in the low nibble to indicate the various error and completion states. LstStatus is guaranteed to be 0 when the request list is submitted to the physical device driver.

DDReserved Reserved for device driver use in tracking request completion. Since the number of requests in the list is variable, this field might be used to point to an auxiliary structure maintained by the device driver.

Each request has a fixed length header, followed by a variable length, command-specific area. The general format of an extended request is shown in the following table:

Field	Length
Request Header	32 BYTES
Command-Specific	BYTE

Extended Commands

The following extended commands, and corresponding command codes, are defined for use in Request Lists:

1Eh	READ
1Fh	WRITE
20h	WRITE VERIFY
21h	READ PREFETCH

The format of the command-specific portion of the request packet format along with details of each command are described in the sections that follow.

Request Header

Each request in the Request List begins with a fixed length header, which has the following format:

Field	Length
Request Length	WORD
Command Prefix=1Ch	BYTE
Command Code	BYTE
Header Offset	DWORD
Request Control	BYTE
Priority	BYTE
Status	BYTE
Error Code	BYTE
Notify Address	DWORD
Hint Pointer	DWORD
DDReserved	DWORD
DDReserved	DWORD
DDReserved	DWORD
Request Length	The offset of the next request. If this is the last request, the value is FFFFH.
Command Prefix	At the same offset in this request header as the command code in the standard OS/2 request header. This byte is always set to EXTENDED REQUEST (1Ch) to allow the physical device driver to maintain one queue for both standard and extended requests, while distinguishing the

two packet types.

Command Code	The request command code. Can have any of the values defined in Extended Commands.														
Header Offset	The offset from the beginning of the Request List header to the header of this request. This field, when subtracted from the address of the header of this request, yields the header of the list. This provides fast access to the control information in the header.														
Request Control	<p>A bit field of control flags as follows:</p> <table><tr><td>Bits 0-3</td><td>Reserved. Must be 0.</td></tr><tr><td>Bit 4</td><td>If set, notify on error only. Indicates that the individual request notification routine, <code>NotifyAddress</code>, should be called immediately in the event that an unrecoverable error occurs in servicing the request.</td></tr><tr><td>Bit 5</td><td>If set, notify on completion. Indicates that the individual notification routine, <code>NotifyAddress</code>, should be called when execution of the request has completed successfully and possibly with a recoverable error. This bit does not indicate notification in the case of an unrecoverable error.</td></tr></table> <p>If bit 4 and bit 5 are clear, the individual request notification routine address is not valid. If bit 4 or bit 5 is set, the notification routine address is valid. Bits 4 and 5 can both be set to indicate notification in case of error or successful completion.</p> <table><tr><td>Bits 6-7</td><td>Reserved. Must be 0.</td></tr></table>	Bits 0-3	Reserved. Must be 0.	Bit 4	If set, notify on error only. Indicates that the individual request notification routine, <code>NotifyAddress</code> , should be called immediately in the event that an unrecoverable error occurs in servicing the request.	Bit 5	If set, notify on completion. Indicates that the individual notification routine, <code>NotifyAddress</code> , should be called when execution of the request has completed successfully and possibly with a recoverable error. This bit does not indicate notification in the case of an unrecoverable error.	Bits 6-7	Reserved. Must be 0.						
Bits 0-3	Reserved. Must be 0.														
Bit 4	If set, notify on error only. Indicates that the individual request notification routine, <code>NotifyAddress</code> , should be called immediately in the event that an unrecoverable error occurs in servicing the request.														
Bit 5	If set, notify on completion. Indicates that the individual notification routine, <code>NotifyAddress</code> , should be called when execution of the request has completed successfully and possibly with a recoverable error. This bit does not indicate notification in the case of an unrecoverable error.														
Bits 6-7	Reserved. Must be 0.														
Priority	<p>A bit field indicating the priority of the request. The following values are currently defined, and others may be added as needed, without notice:</p> <table><tr><td>00h</td><td>Prefetch Requests.</td></tr><tr><td>01h</td><td>Low priority request to be satisfied in the context of servicing other, higher priority requests or when no other work exists (lazy-write).</td></tr><tr><td>02h</td><td>Read ahead, low priority pager I/O.</td></tr><tr><td>04h</td><td>Background synchronous user I/O.</td></tr><tr><td>08h</td><td>Foreground synchronous user I/O.</td></tr><tr><td>10h</td><td>High priority pager I/O.</td></tr><tr><td>80h</td><td>Urgent request; all requests at this priority should be satisfied in a single sweep of the disk; no stopping allowed at cylinders other than those necessary to satisfy requests in this priority. The kernel uses this priority in cases such as an impending power failure or shutdown.</td></tr></table> <p>The kernel or client FSD can request a priority change after the initial submission of the request to the physical device driver by issuing a call to <code>DD _ChgPriority</code>.</p>	00h	Prefetch Requests.	01h	Low priority request to be satisfied in the context of servicing other, higher priority requests or when no other work exists (lazy-write).	02h	Read ahead, low priority pager I/O.	04h	Background synchronous user I/O.	08h	Foreground synchronous user I/O.	10h	High priority pager I/O.	80h	Urgent request; all requests at this priority should be satisfied in a single sweep of the disk; no stopping allowed at cylinders other than those necessary to satisfy requests in this priority. The kernel uses this priority in cases such as an impending power failure or shutdown.
00h	Prefetch Requests.														
01h	Low priority request to be satisfied in the context of servicing other, higher priority requests or when no other work exists (lazy-write).														
02h	Read ahead, low priority pager I/O.														
04h	Background synchronous user I/O.														
08h	Foreground synchronous user I/O.														
10h	High priority pager I/O.														
80h	Urgent request; all requests at this priority should be satisfied in a single sweep of the disk; no stopping allowed at cylinders other than those necessary to satisfy requests in this priority. The kernel uses this priority in cases such as an impending power failure or shutdown.														
Status	A bit field indicating the status of the request. The														

low nibble indicates the completion status of the request, giving the Status byte the following values:

X0h	Not yet queued
X1h	Queued and waiting
X2h	In process
X4h	Done
X8h	Reserved

The high nibble indicates the error status of the request, giving the Status byte the following values:

0Xh	No error
1Xh	A recoverable error has occurred
2Xh	An unrecoverable error has occurred
3Xh	An unrecoverable error has occurred
4Xh	The request was abnormally ended
8Xh	Reserved

High and low nibbles can be set in combination by the physical device driver to indicate combinations of error and status conditions. For example, a code of 12h indicates that a recoverable error has occurred and the request is still in progress. An error condition indicates a valid error code in the ErrorCode field. Status is guaranteed to be 00h when the request is submitted.

Error Code Contains a valid error condition, if an error status is indicated in Status. The following error codes are possible if the error is unrecoverable and are compatible with previous OS/2 error returns:

00h	Write-protect violation
01h	Unknown unit
02h	Device not ready
03h	Unknown command
04h	CRC error
06h	Seek error
07h	Unknown media
08h	Block not found
0Ah	Write fault
0Bh	Read fault
0Ch	General failure
10h	Uncertain media
13h	Invalid parameter

The following error codes are possible, if the error is recoverable:

1Ah	Verify error on write, succeeded after retry
2Ah	Write error, write to mirrored or duplexed drive succeeded
3Ah	Write error on mirrored or duplexed drive; write to primary drive succeeded
1Bh	Read error, corrected using ECC

2Bh Read succeeded after retry
3Bh Read error, recovered from mirrored or duplexed drive

NotifyAddress The address of a notification routine to be called (according to the flags in RequestControl) when the request has completed successfully or unsuccessfully due to error conditions. NotifyAddress is not valid if bits 4 and 5 of RequestControl are clear. NotifyAddress is called with the following parameters:

ES:BX 16:16 Address of the request header

CF Set, if an unrecoverable error has occurred.

The physical device driver is responsible for saving and restoring any registers that must survive the call.

Hint Pointer A 16:16 pointer to a request packet in a Request List. This field can be used when the kernel or the client FSD determines that this request might be best grouped with another request it has already submitted to the physical device driver. The request might have already completed, so the physical device driver must validate that the pointer points to a request on its internal queues. This field is FFFF:FFFFH if it is unused (that is, if a hint is not being passed).

DDReserved Fields reserved for device driver use.

Write/Read/WriteVerify

The format of the request packet for the WRITE, READ, and WRITE VERIFY commands is:

Field	Length
Request Header	32 BYTES
Start Block	DWORD
Block Count	DWORD
BlocksXferred	DWORD
Flags	WORD
SGDescrCount	WORD
Reserved	DWORD
SGDescriptors	ARRAY

Request Header	The fixed length request header.						
Start Block	The starting disk block for the data transfer operation. A disk block is defined as a 512-byte logical disk sector.						
Block Count	The number of 512-byte blocks to be transferred.						
BlocksXferred	The number of 512-byte blocks successfully transferred by the driver. This field is updated before the request and request list notification routines are called and before the Status field is marked as <i>Done</i> .						
Flags	A bit field of command-specific control flags. The following flags have been defined: <table><tr><td>Bit 0</td><td>If set, write through. Defeats any <i>lazy-write</i> caching performed by the physical device driver. Notice that lazy-write, through battery-backed RAM, is permitted, even if this bit is set.</td></tr><tr><td>Bit 1</td><td>If set, cache request on outboard controller cache.</td></tr><tr><td>Bits 2-15</td><td>Reserved, set to 0.</td></tr></table>	Bit 0	If set, write through. Defeats any <i>lazy-write</i> caching performed by the physical device driver. Notice that lazy-write, through battery-backed RAM, is permitted, even if this bit is set.	Bit 1	If set, cache request on outboard controller cache.	Bits 2-15	Reserved, set to 0.
Bit 0	If set, write through. Defeats any <i>lazy-write</i> caching performed by the physical device driver. Notice that lazy-write, through battery-backed RAM, is permitted, even if this bit is set.						
Bit 1	If set, cache request on outboard controller cache.						
Bits 2-15	Reserved, set to 0.						

SGDescrCount	The number of scatter/gather descriptors in the SGDescriptors field.
SGDescriptors	An array of scatter/gather descriptors describing the buffers for data transfer specified by the command.

The File System Driver (FSD) guarantees the following to be true:

$\text{BlockCount} * 512$ equals the sum of the BufferSize fields in SGDescriptors.

In addition, buffers are typically DWORD aligned. The physical device driver should be optimized for this case, but should not rely upon it.

Scatter/Gather Descriptor

READ and WRITE operations use an array of scatter/gather descriptors to describe the buffer space to be used in the operation. This enables transfers of contiguous disk blocks into physically discontinuous, byte-aligned memory blocks. Scatter/gather descriptors have the following format:

Field	Length
BufferPtr	DWORD
Buffer Size	DWORD

BufferPtr	A 32-bit physical pointer to the buffer
Buffer Size	Size of the buffer in bytes

READ PREFETCH

READ PREFETCH is defined to take advantage of a two-tiered disk caching scheme, where the first tier is the file system and the second tier is the controller buffer. This command is optionally supported, if the Read Prefetch bit is set in the VolDescriptor bit field of the VCS for the device. If this bit is set, it is assumed that:

- o The physical device driver manages a cache located on the controller.
- o A Read into controller memory, followed by a Read into system memory, is less expensive (in terms of host CPU utilization) than just a Read into system memory.

If both of these conditions are not met, then the physical device driver does not publish READ PREFETCH capabilities because it is more efficient for the file system to perform *read-ahead* into its own cache. READ PREFETCH commands are the lowest priority requests submitted to the physical device driver through the extended strategy routine and are never serviced prior to other Read/Write requests.

The format of the request packet for READ PREFETCH is:

Field	Length
Request Header	32 BYTES
Start Block	DWORD
Block Count	DWORD
BlocksXferred	DWORD
Flags	WORD
Reserved. Must be 0.	WORD

Request Header	The fixed length request header.
Start Block	The starting disk block for the data prefetch operation. A disk block is defined as a 512-byte logical disk sector.
Block Count	The number of 512-byte blocks to be prefetched.
BlocksXferred	The number of 512-byte blocks successfully prefetched by the

physical device driver. This field is updated before the request and request list notification routines are called and before the Status field is marked as *Done*.

Flags A bit field of command-specific control flags. The following flags have been defined:

Bit 0 If set, hold only until read. The physical device driver retains the data in controller prefetch buffers only until it is read once. This is to prevent redundant caching in the controller.

Bits 1-15 Reserved, set to 0.

Request Control Functions

Request control functions are used by the FSD to manage requests after they have been submitted to obtain advisory information from the physical device driver and to pass advisory information to the physical device driver. Entry points are exported by the physical device driver through the GET DRIVER CAPABILITIES command. Request control functions can be called at interrupt time and cannot block. Request control functions need only preserve segment registers.

The following request control functions are defined:

- o DD_SetFSDInfo
- o DD_ChgPriority
- o DD_SetRestPos
- o DD_GetBoundary

DD_SetFSDInfo

This entry point allows the FSD to inform the physical disk device driver of its FSD_EndOfInt and FSD_AccValidate entry points. The physical disk device driver allows DD_SetFSDInfo to be called exactly once, and ignores subsequent calls.

ENTRY

ES:BX 16:16 pointer to the FSDInfo structure.

EXIT

CF Set, if call was ignored.

The format of the FSDInfo structure is:

Field	Length
Reserved. Must be 0.	DWORD
FSD_EndOfInt	DWORD
Reserved. Must be 0.	DWORD
FSD_AccValidate	DWORD

FSD_EndOfInt The 16:16 entry point of the FSD's FSD_EndOfInt routine. This field is set to 0 if the FSD does not provide an FSD_EndOfInt routine. The entry point is called by the physical device driver when it has completed interrupt processing and after it has called DevHlp_EOI. FSD_EndOfInt takes no parameters and leaves all registers intact.

FSD_AccValidate The 16:16 entry point of the FSD's FSD_AccValidate routine. This field is set to 0 if the FSD does not provide an FSD_AccValidate routine. The entry point is called whenever direct I/O is done through a Category 08h or 09h IOCtl to an HPFS volume or whenever direct I/O is done to the Master Startup (Boot) record through a Category 09h IOCtl.

For Category 09h IOctls, the physical device driver must use the Head, Cylinder, and Sector values passed in the IOctl. These values are used to determine whether the I/O request falls within an HPFS volume, because the unit number in the IOctl represents the entire physical disk and not the logical volume.

The physical device driver should call FSD_AccValidate with:

AL Operation Code
 00 Non-destructive (READ, VERIFY)
 01 Destructive (WRITE, FORMAT TRACK, and so forth)

On return from the FSD:

NC I/O access is allowed.
CF If set, access is denied. The physical device driver (Write-protection violation) to the caller.

DD_ChgPriority

This entry point allows the FSD to notify the physical device driver of a possible change in the priority of a request. HPFS calls this entry point with:

ENTRY

ES:BX Address of the request
AL New priority for the request;

EXIT

CF Set, if request packet not found on any of the physical
 device driver's internal queues

This call is used to change the priority of a previously submitted request. The physical device driver performs whatever resorting of internal queues is necessary, and returns.

The FSD guarantees that the pointer that was passed references a valid request (that is, a request with allowed values in all fields). There is no guarantee that the priority of the request has actually been changed or that the request is still on the internal queues of the driver. If the request has been removed from internal queues, has already been incorporated into internal structures in preparation for service, or has already been serviced, the physical device driver can ignore the requested change.

DD_SetRestPos

This entry point advises the physical device driver of a block to seek when there is no work in the queue. No immediate action is necessary when the call is made. This call is purely advisory and can be ignored by the driver if it is not useful or applicable to the hardware it supports.

ENTRY

AX:BX Block to use as resting point, FFFF:FFFFH, if none
CL Logical Unit Number (A:=0)

EXIT

CF Set, if block is out of range.

The physical device driver updates a static variable, specifying where to rest the heads during idle time. When any seek occurs, either as a result of this call or as the result of I/O requests, the variable is set to FFFFFFFFH. A value of FFFFFFFFH indicates *rest-where-you-end-up*.

This call essentially assumes that there is only one active logical volume serviced by the underlying physical device. *Physical device*, in this context, means mechanical, usually multi-headed, disk arm. If this is not the case, this call should be ignored by the driver.

DD_GetBoundary

This entry point returns the first block number that is greater than the block number specified in the DWORD passed and is past an access time boundary, such as a cylinder. This information can be used by file systems to optimally place file system objects.

ENTRY

AX:BX Reference block number

EXIT

AX:BX Number of first block past access time boundary

CF Set, if block number out of range.

If the physical device driver cannot compute this efficiently, it can precompute this information and retain it internally, or if this is not feasible, it can return (AX:BX) + 1.

I/O Request Block - C Definitions

Following are the I/O request block C language definitions for ADD device support.

```
/*static char *SCCSID = "@(#)iorb.h      6.2 92/02/20";*/
/*****
/* I/O Request Block (IORB) Structures
*****/

/* ASM

        Resolve H2INC references for .INC version of file

        include  iorbtype.inc
*/

/* Typedefs to resolve forward references
typedef struct _IORBH          IORBH;
typedef struct _IORBH          FAR *PIORBH;
typedef struct _IORBH          *NPIORBH;
typedef struct _IORBH          FAR *PIORB;
typedef struct _IORBH          *NPIORB;

typedef struct _DEVICETABLE     DEVICETABLE;
typedef struct _DEVICETABLE     FAR *PDEVICETABLE;
typedef struct _DEVICETABLE     *NPDEVICETABLE;

typedef struct _UNITINFO        UNITINFO;
typedef struct _UNITINFO        FAR *PUNITINFO;
typedef struct _UNITINFO        *NPUNITINFO;

typedef struct _ADAPTERINFO     ADAPTERINFO;
typedef struct _ADAPTERINFO     FAR *PADAPTERINFO;
typedef struct _ADAPTERINFO     *NPADAPTERINFO;

typedef struct _GEOMETRY         GEOMETRY;
typedef struct _GEOMETRY         FAR *PGEOMETRY;
typedef struct _GEOMETRY         *NPGEOMETRY;

typedef struct _SCATGATENTRY     SCATGATENTRY;
```

```

typedef struct _SCATGATENTRY FAR *PSCATGATENTRY;
typedef struct _SCATGATENTRY *NPSCATGATENTRY;

/*-----
/* Interface for calling ADD entry point
/*-----
/* VOID FAR *(ADDEP) (PIORBH);

/*-----
/* IORB Header
/*-----

#define DM_WORKSPACE_SIZE      20
#define ADD_WORKSPACE_SIZE    16

typedef struct _IORBH {          /* IOH

USHORT      Length;             /* IORB length
USHORT      UnitHandle;         /* Unit identifier
USHORT      CommandCode;        /* Command code
USHORT      CommandModifier;    /* Command modifier
USHORT      RequestControl;     /* Request control flags
USHORT      Status;             /* Status
USHORT      ErrorCode;          /* Error code
ULONG       Timeout;            /* Cmd completion timeout(s)
USHORT      StatusBlockLen;     /* Status block length
NPBYTE      pStatusBlock;       /* Status block
USHORT      Reserved_1;         /* Reserved, MBZ
PIORB       pNxtIORB;           /* Pointer to next IORB
PIORB       (FAR *NotifyAddress)(PIORB); /* Notification address
UCHAR       DMWorkSpace[DM_WORKSPACE_SIZE]; /* For use by DM
UCHAR       ADDWorkSpace[ADD_WORKSPACE_SIZE]; /* For use by ADD

} IORBH;

/*-----
/* IORB CommandCode and CommandModifier Codes
/*      Note:  CommandCodes are prefixed by IOCC and CommandModifiers
/*              by IOCM.
/*-----

/*-----
/* +---M=Mandatory support
/* |      O=Optional support

```

```

/* |
/* V      Notes
/*-----

#define IOCC_CONFIGURATION      0x0001 /*
#define IOCM_GET_DEVICE_TABLE  0x0001 /* M
#define IOCM_COMPLETE_INIT     0x0002 /* O
/*-----

#define IOCC_UNIT_CONTROL      0x0002 /*
#define IOCM_ALLOCATE_UNIT     0x0001 /* M
#define IOCM_DEALLOCATE_UNIT   0x0002 /* M
#define IOCM_CHANGE_UNITINFO   0x0003 /* M
/*-----

#define IOCC_GEOMETRY          0x0003 /*
#define IOCM_GET_MEDIA_GEOMETRY 0x0001 /* M
#define IOCM_SET_MEDIA_GEOMETRY 0x0002 /* O (M) >1 media type
#define IOCM_GET_DEVICE_GEOMETRY 0x0003 /* M
#define IOCM_SET_LOGICAL_GEOMETRY 0x0004 /* O (M) CHS addressable
/*-----

#define IOCC_EXECUTE_IO        0x0004 /*
#define IOCM_READ               0x0001 /* M
#define IOCM_READ_VERIFY        0x0002 /* M
#define IOCM_READ_PREFETCH      0x0003 /* O
#define IOCM_WRITE              0x0004 /* M
#define IOCM_WRITE_VERIFY        0x0005 /* M
/*-----

#define IOCC_FORMAT            0x0005 /*
#define IOCM_FORMAT_MEDIA        0x0001 /* O (M) If HW requires
#define IOCM_FORMAT_TRACK        0x0002 /* O (M) If HW requires
#define IOCM_FORMAT_PROGRESS     0x0003 /* O
/*-----

#define IOCC_UNIT_STATUS       0x0006 /*
#define IOCM_GET_UNIT_STATUS    0x0001 /* M
#define IOCM_GET_CHANGELINE_STATE 0x0002 /* O (Mandatory for diskette)
#define IOCM_GET_MEDIA_SENSE    0x0003 /* O (Mandatory for diskette)
#define IOCM_GET_LOCK_STATUS    0x0004 /* M
/*-----

#define IOCC_DEVICE_CONTROL    0x0007 /*
#define IOCM_ABORT              0x0001 /* O (M) SCSI
#define IOCM_RESET              0x0002 /* O (M) SCSI
#define IOCM_SUSPEND            0x0003 /* O (M) Floppy driver
#define IOCM_RESUME             0x0004 /* O (M) Floppy driver
#define IOCM_LOCK_MEDIA         0x0005 /* M (O) Fixed media only
#define IOCM_UNLOCK_MEDIA       0x0006 /* M (O) Fixed media only
#define IOCM_EJECT_MEDIA        0x0007 /* M (O) SCSI and Floppy Driver

```

```

/*-----
#define IOCC_ADAPTER_PASSTHRU    0x0008  /*
#define IOCM_EXECUTE_SCB        0x0001  /* 0
#define IOCM_EXECUTE_CDB        0x0002  /* 0 (M) SCSI adapters
/*-----

#define MAX_IOCC_COMMAND    IOCC_ADAPTER_PASSTHRU

/*-----
/* Status flags returned in IORBH->Status
/*-----
#define IORB_DONE            0x0001  /* 1=Done, 0=In progress
#define IORB_ERROR          0x0002  /* 1=Error, 0=No error
#define IORB_RECOV_ERROR    0x0004  /* Recovered error
#define IORB_STATUSBLOCK_AVAIL 0x0008  /* Status block available

/*-----
/* Error Codes returned in IORBH->ErrorCode
/*-----

#define IOERR_RETRY            0x8000

#define IOERR_CMD              0x0100
#define IOERR_CMD_NOT_SUPPORTED IOERR_CMD+1
#define IOERR_CMD_SYNTAX      IOERR_CMD+2
#define IOERR_CMD_SGLIST_BAD  IOERR_CMD+3
#define IOERR_CMD_SW_RESOURCE IOERR_CMD+IOERR_RETRY+4
#define IOERR_CMD_ABORTED     IOERR_CMD+5
#define IOERR_CMD_ADD_SOFTWARE_FAILURE IOERR_CMD+6
#define IOERR_CMD_OS_SOFTWARE_FAILURE IOERR_CMD+7

#define IOERR_UNIT            0x0200
#define IOERR_UNIT_NOT_ALLOCATED IOERR_UNIT+1
#define IOERR_UNIT_ALLOCATED   IOERR_UNIT+2
#define IOERR_UNIT_NOT_READY   IOERR_UNIT+3
#define IOERR_UNIT_PWR_OFF     IOERR_UNIT+4

#define IOERR_RBA              0x0300
#define IOERR_RBA_ADDRESSING_ERROR IOERR_RBA+IOERR_RETRY+1
#define IOERR_RBA_LIMIT        IOERR_RBA+2
#define IOERR_RBA_CRC_ERROR     IOERR_RBA+IOERR_RETRY+3

#define IOERR_MEDIA            0x0400

```

```

#define IOERR_MEDIA_NOT_FORMATTED      IOERR_MEDIA+1
#define IOERR_MEDIA_NOT_SUPPORTED      IOERR_MEDIA+2
#define IOERR_MEDIA_WRITE_PROTECT     IOERR_MEDIA+3
#define IOERR_MEDIA_CHANGED           IOERR_MEDIA+4
#define IOERR_MEDIA_NOT_PRESENT       IOERR_MEDIA+5

#define IOERR_ADAPTER                 0x0500
#define IOERR_ADAPTER_HOSTBUSCHECK    IOERR_ADAPTER+1
#define IOERR_ADAPTER_DEVICEBUSCHECK IOERR_ADAPTER+IOERR_RETRY+2
#define IOERR_ADAPTER_OVERRUN        IOERR_ADAPTER+IOERR_RETRY+3
#define IOERR_ADAPTER_UNDERRUN       IOERR_ADAPTER+IOERR_RETRY+4
#define IOERR_ADAPTER_DIAGFAIL       IOERR_ADAPTER+5
#define IOERR_ADAPTER_TIMEOUT        IOERR_ADAPTER+IOERR_RETRY+6
#define IOERR_ADAPTER_DEVICE_TIMEOUT IOERR_ADAPTER+IOERR_RETRY+7
#define IOERR_ADAPTER_REQ_NOT_SUPPORTED IOERR_ADAPTER+8
#define IOERR_ADAPTER_REFER_TO_STATUS IOERR_ADAPTER+9
#define IOERR_ADAPTER_NONSPECIFIC    IOERR_ADAPTER+10

#define IOERR_DEVICE                 0x0600
#define IOERR_DEVICE_DEVICEBUSCHECK  IOERR_DEVICE+IOERR_RETRY+1
#define IOERR_DEVICE_REQ_NOT_SUPPORTED IOERR_DEVICE+2
#define IOERR_DEVICE_DIAGFAIL       IOERR_DEVICE+3
#define IOERR_DEVICE_BUSY            IOERR_DEVICE+IOERR_RETRY+4
#define IOERR_DEVICE_OVERRUN        IOERR_DEVICE+IOERR_RETRY+5
#define IOERR_DEVICE_UNDERRUN       IOERR_DEVICE+IOERR_RETRY+6
#define IOERR_DEVICE_RESET          IOERR_DEVICE+7
#define IOERR_DEVICE_NONSPECIFIC    IOERR_DEVICE+8

/*-----
/* Request Control flags in IORBH->RequestControl
/*-----

#define IORB_ASYNC_POST      0x0001  /* Asynchronous post enabled
#define IORB_CHAIN           0x0002  /* IORB chain link enabled
#define IORB_CHS_ADDRESSING  0x0004  /* CHS fmt addr in RBA field
#define IORB_REQ_STATUSBLOCK 0x0008  /* Obtain status block data
#define IORB_DISABLE_RETRY   0x0010  /* Disable retries in ADD

/*****
/* ADAPTER CONFIGURATION IORB (for IOCC_CONFIGURATION)
/*****
typedef struct _IORB_CONFIGURATION { /* IOCFG

```

```

    IORBH          iorbh;          /* IORB header
    DEVICETABLE far *pDeviceTable;  /* Far pointer to adapt. dev. table
    USHORT         DeviceTableLen;  /* Length of adapter device table
} IORB_CONFIGURATION, FAR *PIORB_CONFIGURATION, *NPIORB_CONFIGURATION;

/* Adapter device table returned by GET_DEVICE_TABLE
typedef struct _DEVICETABLE {      /* IODT

    UCHAR          ADDLevelMajor;   /* ADD major support level
    UCHAR          ADDLevelMinor;   /* ADD minor support level
    USHORT         ADDHandle;       /* ADD handle
    USHORT         TotalAdapters;   /* Number of adapters supported
    NPADAPTERINFO pAdapter[1];      /* Array of adapter info pointers

} DEVICETABLE, FAR *PDEVICETABLE;

/*-----
/* Current ADD Level for DEVICETABLE->AddLevelMajor, AddLevelMinor
/*-----

#define ADD_LEVEL_MAJOR      0x01
#define ADD_LEVEL_MINOR      0x00

typedef struct _UNITINFO {        /* IOUI

    USHORT         AdapterIndex;    /* nth adapter this driver
    USHORT         UnitIndex;       /* nth unit on this card
    USHORT         UnitFlags;       /* Unit flags
    USHORT         Reserved;        /* Reserved; must be 0
    USHORT         UnitHandle;      /* Assigned by ADD or filter
    USHORT         FilterADDHandle; /* Handle of filter ADD 0=None

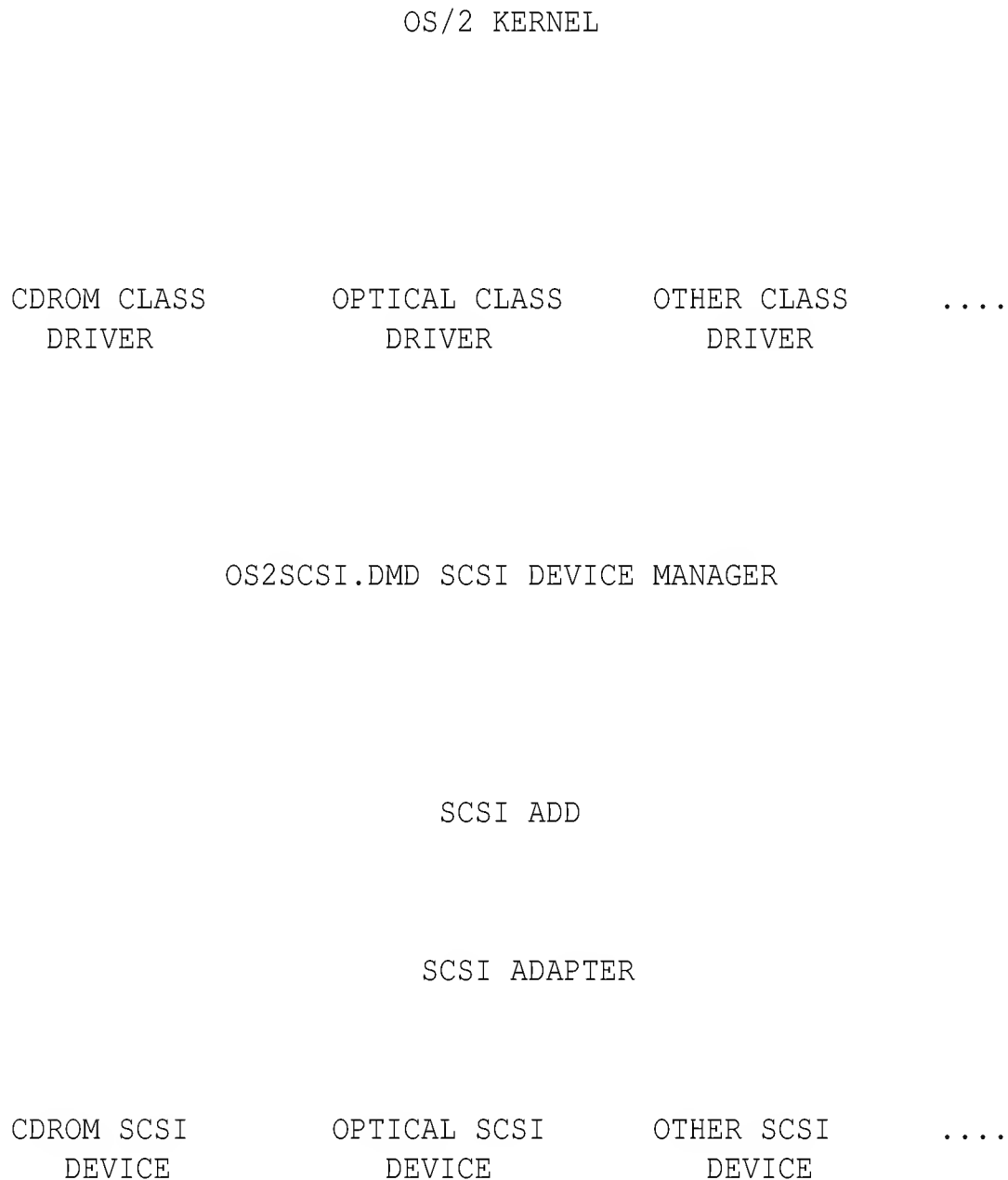
```

OS/2 SCSI Device Driver Interface Specification

This appendix describes the high-level interface for the SCSI device driver for the OS/2 operating system. For completeness, all functions are listed; however, functions that are not implemented are so indicated. Some of the internal specifications of the device driver have not been included here so that this document can be externally distributed to vendors wanting to write device drivers to the SCSI device driver interface.

Introduction

The SCSI driver is the lower half of a *split* model for OS/2 SCSI device drivers. The SCSI driver drives the SCSI adapter through the SCSI adapter device driver as shown in the following figure.



The diagram illustrates the relationship between the device drivers and their interaction with other components of the system.

The *split device driver* model uses the principles of code *layering* to facilitate development and maintenance of new SCSI device drivers. The provision of common functions in the SCSI driver also reduces memory requirements. Performance is enhanced because the SCSI driver centralizes control of the SCSI channel, thus reducing contention. Only one interrupt handler is registered for all the SCSI peripheral devices.

A split device driver model has been used by IBM for all the SCSI devices except the SCSI fixed disks, which use the OS/2 DASD Manager. The *device-class driver* is the upper-level driver, and the SCSI driver is the lower-level driver. The device-class driver does not interact directly with the SCSI adapter or the SCSI device. The device-class driver sends commands to the SCSI device manager, which in turn sends commands to the device using the IORB ADD interface.

The device-class driver looks very much like an OS/2 device driver. It maps an OS/2 request into an *SCB*, or a chain of SCBs, and passes the request immediately to the SCSI driver. The SCSI driver handles all queuing and interrupts and insulates the device-class driver from the procedural details of managing adapter hardware. The device-class driver requests a service, like *Transfer SCB*, from the SCSI device manager. When control is returned to the device-class driver, the called service is complete. If an error occurred, the termination status block (TSB) might contain error information. In addition, sense data might have been returned.

Naming Conventions

- ***SCSI Driver***

The file name for the SCSI driver is *OS2SCSI.DMD*. The IDC entry point for the SCSI driver can be obtained from the **AttachDD** DevHlp function by using the name *SCSI-02\$* as the device driver name parameter.

- ***IBM Device-Class Drivers***

Current device driver names used by IBM are:

OPTICAL.SYS	Read/Write optical device driver
-------------	----------------------------------

Note: Naming conflicts are possible, so try to choose unique names for your device-class drivers. In a SCSI environment, different vendor devices for the same SCSI device class can be present in one system.

- ***Message Files***

The IBM-reserved message file name for device drivers that have been developed internally is DEV002.MSG. Independent vendors *must not* use DEV002.MSG for their message files, because if they do, one of those message files could be destroyed during the user installation process. It is suggested that vendors choose a unique message file name by embedding part of their logo or company name in the file name; that would eliminate the possibility of having different vendor devices with the same message file name installed in the same system.

Generic IOCTL Request

The input to the SCSI driver is a generic IOCTL request packet, pointed to by the ES:BX register pair.

Note: It is not the intention of the IOCTL definitions in the SCSI.SYS specification to provide an application level programming interface to SCSI devices. The intention of the IOCTL definitions is to allow client drivers at their Ring 3 INIT time to communicate with the OS2SCSI.DMD which is running at a higher privilege level than the installable driver.

A client driver may choose to provide application level services. However, all responsibility for locking and removing LDT references from the IOCTL/SCB/TSB structures would rest with the client driver *not* OS2SCSI.DMD.

OS/2 SCSI Services

The SCSI driver supports the following requests:

- o Read Device Parameters
- o Reset/Initialization
- o Enable Adapter Cache
- o Disable Adapter Cache
- o Return Adapter Cache Status
- o Set Device Timeout
- o Read Device Timeout
- o Transfer SCB
- o Deallocate Device
- o Allocate Device
- o Return Peripheral Type Count
- o Abort

Read Device Parameters

This function returns some information about the device. The logical unit number (LUN) is required if a Send Other command is used.

Input Parameter Structure

Field	Length
Device Handle	WORD

FUNCTION CATEGORY : 80h
FUNCTION CODE : 43h

This function requires a device handle to be passed in the request. The device must be allocated by the device-class driver before calling this function. The function category and function code are to be set up as shown above.

Field Name	Length
Device Key Index	WORD
SCB Architecture Card Comp. Level	BYTE
Adapter Index	BYTE
Device Flags	WORD
Logical unit number (LUN)	BYTE
Physical unit number (PUN)	BYTE

Adapter Index

contains the adapter number for the SCSI adapter.

Device Flags

Bit 4 0 = Adapter cache not supported.
 1 = Adapter cache supported.

Bit 1 0 = Device power ON.
 1 = Device power OFF.

Bit 0 0 = Device is not defective.
 1 = Device is defective.

Reset/Initialize

This function results in a reset message being issued to the physical device.

Input Parameter Structure

Field	Length
Device Handle	WORD
Sense Data Size	WORD

FUNCTION CATEGORY : 80h
FUNCTION CODE : 45h

This function requires a device handle to be passed in the request. The device must be allocated by the device-class driver before calling this function. The function category and function code must be set up as shown above.

Data Buffer

This function does not require a data buffer. Status is returned in the *Status* field of the request packet.

Enable Adapter Cache

This function enables the adapter cache capability for all subsequent commands to this device.

Input Parameter Structure

Field	Length
Device Handle	WORD

FUNCTION CATEGORY : 80h
FUNCTION CODE : 4Dh

This function requires a device handle to be passed in the request. The device must be allocated by the device-class driver before calling this function. The function category and function code must be set up as shown above.

Data Buffer

This function does not require a data buffer. Status is returned in the *Status* field of the request packet.

Disable Adapter Cache

This function disables the adapter cache capability for subsequent commands to the specified device.

Input Parameter Structure

Field	Length
Device Handle	WORD

FUNCTION CATEGORY : 80h
FUNCTION CODE : 4Eh

This function requires a device handle to be passed in the request. The device must be allocated by the device-class driver before calling this function. The function category and function code must be set up as shown above.

Data Buffer

This function does not require a data buffer. Status is returned in the *Status* field of the request packet.

Return Adapter Cache Status

This function returns the adapter cache status for the specified device.

Input Parameter Structure

Field	Length
Device Handle	WORD

FUNCTION CATEGORY : 80h

FUNCTION CODE : 4Fh

This function requires a device handle to be passed in the request. The device must be allocated by the device-class driver before calling this function. The function category and function code must be set up as shown above.

Field Name	Length
Adapter Cache Status	BYTE

Adapter Cache Status : 00H Enabled
 01H Disabled

Set Device Timeout

This function sets the timeout value for this device.

Input Parameter Structure

Field	Length
Timeout Value	DWORD

FUNCTION CATEGORY : 80h

FUNCTION CODE : 50h

This function requires a device handle and a timeout value to be passed in the request. The timeout value is in milliseconds. The device must be allocated by the device-class driver before calling this function. The function category and function code must be set up as shown above.

Data Buffer

This function does not require a data buffer. Status is returned in the *Status* field of the request packet.

Read Device Timeout

This function returns the current timeout value for this device.

Input Parameter Structure

Field	Length
Device Handle	WORD

FUNCTION CATEGORY : 80h

FUNCTION CODE : 51h

This function requires a device handle to be passed in the request. The device must be allocated by the device-class driver before calling this function. The function category and function code must be set up as shown above.

Field Name	Length
Timeout Value	DWORD

The timeout value is in milliseconds.

Transfer SCB

This function causes an SCB or a chain of SCBs to be sent to the adapter.

Field Name	Length
Device Handle	WORD
Sense Data Size	WORD
Physical Pointer to SCB	DWORD
Logical Pointer to SCB Chain Header	DWORD
Flags	BYTE

FUNCTION CATEGORY : 80h
FUNCTION CODE : 52h

This function requires a device handle to be passed in the request. The device must be allocated by the device-class driver before calling this function. The function category and function code are to be set up as shown above.

Flags

Bit 0 = 0 Normal Length SCB
 1 Long SCB

A normal length SCB is used to send generic SCSI commands to a device. The long SCB is used to send a vendor-unique SCSI command embedded in the SCB.

Data Buffer

If an error occurs, the data buffer might contain sense data; the return code indicates whether the sense data is valid. A termination status block also might be returned.

SCB Chain Header

+00h	Reserved
+02h	Logical Pointer to next SCB Chain Header
+06h	Reserved
+08h	Reserved
+0Ah	Logical Pointer to TSB
+0Eh	Reserved
+10h	

SCB

Immediately

Follows

the

Chain

Header

See Subsystem Control Blocks for a description of the SCB architecture.

Allocate Device

This function allocates a SCSI peripheral device and returns the device handle that will be used to access the device.

Input Parameter Structure

Field	Length
Device Peripheral Type	BYTE
Device Type Flags	BYTE
Nth Available	WORD

FUNCTION CATEGORY : 80h

FUNCTION CODE : 55h

This function requires a device type, device type flags, and Nth available device to be passed in the request. The device type flags define the *removable media indicator*. The most significant bit of the device type flags set indicates that the media is removable. The Nth available is the Nth device in the device type group. If Nth available is 0, the next available device is returned.

o SCSI Device Types

Direct Access	0x00
Sequential Access	0x01
Printer	0x02
Processor	0x03
Write Once/Read Many	0x04
CD-ROM	0x05
Scanner	0x06
Optical Memory	0x07
Medium Changer	0x08
Communications	0x09

Data Buffer

Field Name	Length
Device Handle	WORD

Device Handle

Returned to the caller.

Deallocate Device

This function deallocates the SCSI Peripheral Device assigned to this device handle.

Input Parameter Structure

Field	Length
Device Handle	WORD

FUNCTION CATEGORY : 80h
FUNCTION CODE : 54h

This function requires a device handle to be passed in the request. The device must be allocated by the device-class driver before calling this function. The function category and function code must be set up as shown above.

Data Buffer

This function does not require a data buffer. Status is returned in the *Status* field of the request packet.

Return Peripheral Type Count

This function returns a count of the number of devices of a particular type that are detected.

Input Parameter Structure

Field	Length
Device Peripheral Type	BYTE
Device Type Flags	BYTE

FUNCTION CATEGORY : 80h

FUNCTION CODE : 56h

This function requires a device type and device type flags to be passed in the request. The device type flags define the removable media indicator. The most significant bit of the device type flags set indicates that the media is removable. Function category and function code must be set up as shown above.

Field Name	Length
------------	--------

Count of Device Type WORD Requested	
-------------------------------------	--

Count of Device Type Requested

Returned when the request is completed successfully.

Send Abort

This function causes an abort request to be sent to the device.

Input Parameter Structure

Field	Length
Device Handle	WORD
Sense Data Size	WORD
Reserved	DWORD

FUNCTION CATEGORY : 80h

FUNCTION CODE : 57h

Data Buffer

This function does not require a data buffer. Status is returned in the *Status* field of the request packet.

Return Codes

The following table describes return code bit categories.

Bit Numbers	Category
15	ERROR
10 - 14	RESERVED
9	BUSY
8	DONE
7	SCSI ERROR
0 - 6	ERROR CODE (when Bit 15 = 1)

The following table describes bit descriptions.

Bit	Description
07	SCSI Driver-Specific Error
08	Operation Complete
15	Request Completed with Error

The following table describes SCSI error codes.

Error Code	Description
00h	Device Error (Sense Data Returned)
01h	Timeout Error

02h	Unusual Wakeup Error
03h	DevHlp Error
04h	Request Block Not Available
05h	Maximum Device Support Exceeded
06h	Interrupt Level Not Available
07h	Device Not Available
08h	More IRQ Levels than Adapters
09h	Device Busy
0Ah	Request Sense Failed
0Bh	Adapter Cache Not Supported

The SCSI device driver can return any of the standard OS/2 device driver return codes as well as the specific error codes listed above.

If Bit 15 is set, Bits 0 - 6 contain an error code. If, in addition, Bit 7 is set, the error code in Bits 0 - 6 is one of the SCSI device driver-specific error codes from the table. Otherwise, it is a standard OS/2 device driver error code, such as `unknown_command` or `invalid_parameter`.

The DONE bit always is set by the SCSI device driver so that a successful return code is hex *0100*, not 0.

At initialization time, the returned status is OR'd with hex *FF00* by the kernel.

Error Recovery Procedure

The SCSI device driver will not perform any error recovery on the SCSI adapter. The SCSI adapter will not be allocated and, therefore, no error recovery procedure is followed.

If a Check Condition is detected, the SCSI device driver will request sense data from the device and return it to the device-specific driver if successful. A return code of hex *xx80* indicates that sense data has been returned.

Device-Class Driver Model

The device-class driver model is described briefly here to assist in the design of a device-class driver.

Overview

The device-class driver receives OS/2 request packets from the kernel. It is responsible for mapping the received request to a generic IOCTL request to be passed to the SCSI device driver. When a request from the kernel results in sending a Transfer SCB command to the SCSI driver, the device-class driver allocates the SCB chain header and formats the SCB and the SCB chain header. The TSB also must be allocated. When a request from the kernel results in multiple Transfer SCBs, the device-class driver chains the SCBs and sends only one Transfer SCB command to the SCSI driver. This achieves better performance and guarantees that requests are processed sequentially.

The device-class driver calls the SCSI driver to send the request to the device. The SCSI driver returns to the device-class driver after the request is completed. When a Transfer SCB request completes with an error, the SCSI driver performs a Request Sense command to the device to obtain sense data. The sense data is passed back to the caller in the data buffer area of the generic IOCTL request packet. The device-class driver might take some error-recovery steps at this point or return to the kernel, passing the return code from the device.

Initialization Routine

This routine is called when the device-class driver is first loaded into the system. This routine performs all initialization required for the device-class driver and the device. At Init time, all calls to the SCSI driver are made through the DosDevIoctl interface. Typically, initialization performs the functions in the following list:

1. Performs a return peripheral device count to determine the count of devices attached.
2. Allocates the device.
3. Queries the device to determine whether it is supported.
4. Sets the return code in the request block.
5. Returns the offsets for the end of the code and data segments.

Strategy Routine

This routine receives requests from the kernel at task time. It builds a generic IOCTL request packet and sends it to the SCSI driver through the IDC entry point.

The generic IOCTL request contains the following parameters:

- FUNCTION CATEGORY
80h
- FUNCTION CODE
Represents function to be performed by the SCSI driver.
- PARAMETER BUFFER ADDRESS
Contains a pointer to the parameters required for the function to be performed.
- DATA BUFFER ADDRESS
Contains a pointer to the data buffer where returned data is stored.

Interrupt Handler

An interrupt handler is not required for the device-class driver. All interrupts from the SCSI peripheral devices are handled by the SCSI driver.

DMA Data Structures

All data structures that will be accessed by the *DMA* must be locked into memory before calling OS2SCSI.DMD. These data structures include the following:

- o SCB chain header and SCBs
- o Scatter/gather list
- o Scatter/gather data areas
- o TSB
- o Sense data area
- o User data areas.

Subsystem Control Blocks

The SCB commands relieve the system of transferring command blocks to the adapter through programmed input and output. The SCB specifies the desired command and associated parameters. An SCB must begin on a doubleword boundary, and any address translations, from virtual to physical, must be performed by system software before the SCB pointer is loaded into the command interface registers. If 80386 virtual page mode is being used, system software must also ensure that the SCB, data buffers, and termination status block (TSB) areas are locked into memory. The SCB specifies the desired command and associated parameters. When the SCB (or chain of SCBs) has been performed by the adapter, an interrupt request is issued to the system. The adapter presents only one interrupt request at a time to the system.

The following figure shows the format of the subsystem control block as it applies to device-related commands.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
<---- Command Dependent ---> ND NS C5 C4 C3 C2 C1 C0																Command Word
RD	ES	RE	PT	0	SS	BB	0	0	0	0	0	0	0	0	CH	Enable Word
<----- Least Significant Word ----->																Logical Block
<----- Most Significant Word ----->																Address
<----- Least Significant Word ----->																System Buffer
<----- Most Significant Word ----->																Address
<----- Least Significant Word ----->																System Buffer
<----- Most Significant Word ----->																Byte Count
<----- Least Significant Word ----->																Termination Status
<----- Most Significant Word ----->																Block Address
<----- Least Significant Word ----->																Optional SCB Chain
<----- Most Significant Word ----->																Address
<----- Number of Blocks ----->																Block Count
<----- Block Size ----->																Block Length

A command is encoded in the first word of the SCB. The setting of bits 15 - 8 of the first word depends on the specific command identified in bits 5 - 0. If bit 7 (ND) of this word is set to 1, the adapter will not disconnect the target device during command execution. If bit 6 (NS) of this word is set to 1, the adapter will not send any Synchronous Data Transfer Request messages to the

target device.

The second word of the SCB is used to enable options that are used to modify a specified command, as shown in the following table.

Bit	Symbol	Function
15	RD	Input/Output Control
14	ES	Report TSB Status Only on Error
13	RE	Retry Enable
12	PT	Pointer to List
10	SS	Suppress Exception Short
9	BB	Bypass Buffer
8 - 1		Reserved
0	CH	Chain on No Error

Bit 15 (RD) When this bit is set to 1, the adapter transfers data from the SCSI device or adapter into system memory (read). When this bit is set to 0, the adapter transfers data from system memory to the SCSI device or adapter (write).

Bit 14 (ES) When this bit is set to 1, the TSB is transferred to memory only if an error (Interrupt ID = C) is detected. When this bit is set to 0, the TSB is always transferred.

Note: This bit should always be set to 1, unless the command requires the TSB when no error occurs; command performance is degraded by unnecessarily writing to memory.

Bit 13 (RE) When this bit is set to 1, the adapter automatically retries certain operations that fail. This bit may be set to 0 by diagnostic programs to enhance fault isolation. Normally, this bit should be set to 1. See Word 1 - Retry Counts for more information.

- Bit 12 (PT) When this bit is set to 1, it allows a single command to write data to or read data from several different areas in memory (buffers) as specified in a list. This list contains up to 16 pairs of values, each pair being a 32-bit address and its related 32-bit count. In the SCB, the system buffer address field contains the address of the list, and the system buffer byte count field contains the length of the list in bytes.
- Bit 10 (SS) When this bit is set to 1, it allows the amount of data transferred on a read operation to be shorter than the system buffer byte count, specified in the SCB, without generating an error.
- Bit 9 (BB) When set to 1, this bit forces the adapter to transfer data directly from the SCSI device and not from a copy in the cache. Some buffer maintenance may still be performed by the adapter.
- Bits 8 - 1 These bits are reserved.
- Bit 0 (CH) This bit selects the type of chaining condition used in command block transfers. When it is set to 0, chaining is disabled. When command blocks are chained, the SCB must contain the 32-bit address of the next SCB. When this bit is set to 1, chaining will occur if the SCB ends with no error.

System Interface

The following is a list of supported SCBS commands.

Command Type	Command	Hex Code
SCB	Device Inquiry	0B
SCB	Format Unit	16
SCB	Get Command Complete Status	07
SCB	Read Data	01
SCB	Read Device Capacity	09
SCB	Read Prefetch	31
SCB	Read Verify	03
SCB	Reassign Block	18
SCB	Request Sense	08
SCB	Send Other SCSI Command (SCSI CDB)	1F
SCB	Write Data	02
SCB	Write with Verify	04

Note: The hex code represents bits 5 - 0 of the first command word.

The adapter maintains a Command Complete status block for each of the command blocks. The command blocks are updated at the completion of each command. This command status block can be obtained by using the Get Command Complete Status Block command. See Command Complete Status Block.

The format for each command is given following the associated command.

Device Inquiry

Through the SCB Device Inquiry command, the system determines which SCSI devices are attached to the adapter, and specific information about those devices. When the Device Inquiry data block has been transferred, the adapter interrupts the system. Because the length of the returned data block is device-dependent, the system should specify the amount of data to be returned. If this is not known, then the system should specify the maximum value (255) and set the suppress short exception (SS) bit to 1. After the Device Inquiry data block is transferred to the specific address, the adapter interrupts the system to indicate that the command is complete.

If a SCSI device is not attached at the assigned physical SCSI address, the command-completed-with-failure interrupt will be returned in the Interrupt Status register. The Command Complete status will indicate selection time-out. If the SCSI logical unit number is not supported by an attached SCSI physical unit, the device type in the Device Inquiry data block is set to hex 7F by the SCSI physical device.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	ND	NS	0	0	1	0	1	1	Device Inquiry
1	ES	RE	0	0	SS	1	0	0	0	0	0	0	0	0	CH	Enable Word
<-----			Reserved			----->										
<-----			Reserved			----->										
<-----			Least Significant Word										----->		System Buffer	
<-----			Most Significant Word										----->		Address	
<-----			Least Significant Word										----->		System Buffer	
<-----			Most Significant Word										----->		Byte Count	
<-----			Least Significant Word										----->		Termination Status Block	
<-----			Most Significant Word										----->		Address	
<-----			Least Significant Word										----->		Optional SCB Chain	
<-----			Most Significant Word										----->		Address	
<-----			Reserved										----->			
<-----			Reserved										----->			

Device Inquiry Data Block

Byte	7	6	5	4	3	2	1	0	Remarks
0	<Peripheral Device Type>								Major Type
1	RMB <- Type Qualifier ->								Removable Media Bit
2	<ISO> <-ECMA-> <-ANSI->								Standards Compliance
3	<----- Reserved ----->								
4	<- Additional Length -->								# Of Bytes (N-4)

5	<-- Additional Data --->								Additional
	.								Inquiry
N	<-- Additional Data --->								Data

ECMA - European Computer Manufacturer's Association
ISO - International Standards Organization

For more information about the Device Inquiry data block, refer to the American National Standards Institute SCSI Standard X3.131-1986.

Format Unit

This SCB command is used to format a storage device. Formatting the storage device destroys all data. The device performs defect management as specified in the command. Bits within the command specify the source of the defect list and the use and disposition of any defect list on the device. Because all device data is considered erased, any cache data for the device is cleared.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	ND	NS	0	1	0	1	1	0	Format Unit
0	ES	RE	0	0	0	1	0	0	0	0	0	0	0	0	CH	Enable Word
<----- Reserved ---->								0	0	0	FD	CL	0	0	0	Modifier Bits
<----- Interleave Factor ----->																Interleave
<----- Least Significant Word ----->																System Buffer
<----- Most Significant Word ----->																Address
<----- Least Significant Word ----->																Defect List
<----- Most Significant Word ----->																Byte Count
<----- Least Significant Word ----->																Termination Status
<----- Most Significant Word ----->																Block Address
<----- Least Significant Word ----->																Optional SCB Chain
<----- Most Significant Word ----->																Address
<----- Number of Blocks ----->																Block Count
<----- Block Size ----->																Block Length

The interleave factor used during the format operation is specified in the control block. An interleave factor of 0 selects the device default. A factor of 1 selects sequential numbering of logical blocks. All other factor values are device dependent.

Modifier bits select options to be used during formatting and are defined as follows:

FD **Format Data:** When this modifier bit is set to 1, the system supplies a defect list for the format operation. The structure of the list depends on the device being formatted. The system buffer address points to the defect list; the length is specified in the byte count. If this bit is set to 0, no defect list is transferred to the device.

Note: Not all SCSI devices support the transfer of a defect list.

CL **Complete List:** If the defect list is supplied, this bit determines whether the supplied defect list is in addition to, or replaces, the defect list already in the device. If the bit is set to 1, any previous defect list is replaced.

Note: Only a defect list in the following block format is supported by the adapter. See the ANSI SCSI Standard or specific device specification for more information.

Byte	Defect List Header								Remarks
	7	6	5	4	3	2	1	0	
0	<----- Reserved ----->								
1	<----- Reserved -BF-->								
2	<----- High Byte ---->								
3	<----- Low Byte ----->								
	Defect Descriptors								
4	<----- High Byte ---->								First
5	<----->								Defective Block
6	<----->								Address
7	<----- Low Byte ----->								
	.								
	.								
	.								
	<----- High Byte ---->								Last
	<----->								Defective Block
	<----->								Address
N	<----- Low Byte ----->								

BF **Background Format:** When this bit is set to 1, the device performs a background format. If the device supports this option, it checks the format of the command, then returns a command status indicating good status, and starts the format operation. If the device does not support the option, it may return a command status block indicating a check condition.

Commands received before completing the background format are returned with a command status block indicating a check condition. The Request Sense command returns a sense key indicating that the device is not ready and returns an additional sense code indicating that a Format operation is in progress. The Request Sense data block also shows the percentage of the format completed.

Get Command Complete Status

This SCB command requests the Command Complete status block for the last command executed on a specified device. When the status block is transferred to the system, the adapter generates an interrupt and updates the Interrupt Status register.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	0	0	0	0	0	1	1	1	Get Command Status
1	ES	RE	0	0	0	1	0	0	0	0	0	0	0	0	CH	Enable Word
<-----			Reserved			----->										
<-----			Reserved			----->										
<-----			Least Significant Word			----->										System Buffer
<-----			Most Significant Word			----->										Address
0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	System Buffer
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Byte Count
<-----			Least Significant Word			----->										Termination Status Block
<-----			Most Significant Word			----->										Address
<-----			Least Significant Word			----->										Optional SCB Chain
<-----			Most Significant Word			----->										Address
<-----			Reserved			----->										
<-----			Reserved			----->										

Command Complete Status Block

The command complete status block is returned to the location specified in the system buffer address field of the Get Command Complete Status command. It contains the status of the last command to a device. It is unchanged until another command is issued to that device or until a reset occurs.

An optional termination status block is returned automatically whenever an error occurs. This allows command complete status to be returned for error recovery.

The command complete status block and termination status block contain the same information.

Word	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	<----- SCB End Status Word ----->																SCB Status
1	<----- Retry Counts ----->																Retry Counts
2	<----- Least Significant Word ----->																Residual Byte
3	<----- Most Significant Word ----->																Count
4	<----- Least Significant Word----->																Scatter/Gather List
5	<----- Most Significant Word ----->																Element Address
6	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	Device Dependent Status Length
7	<-- Command Status -->								<---- Device Status ---->								Command Device Status
8	<-- Command Error --->								<---- Device Error ----->								Error Codes
9	<----- Reserved ----->																
A	<----- Cache Information Word ----->																
B	<----- Least Significant Word ----->																Last SCB Address:
C	<----- Most Significant Word ----->																Processed

Word 0 - Subsystem Control Block End Status

Bit	Function
15 - 13	Reserved
12	Major Exception Occurred
11	Device Not Initialized
10	Reserved
9	Device Dependent Status Available
8	Additional Status Available
7	SCB Interrupt Queued
6	SCB Halted (Error/End Chain)
5	Long Record Exception
4	SCB Specification Check
3	SCB Rejected
2	Invalid Command Rejected
1	Short Record Exception
0	SCB Ended (No Error)

Note: The function indicated is true when the value of the bit is one. Reserved bits are undefined.

Word 1 - Retry Counts

Bit	Function
15	Adapter Retry Invoked
14 - 6	Reserved
5	System Interface Check Retry
4 - 0	Reserved

Words 2 and 3 - Residual Byte Count

These words contain the number of bytes that were not transferred.

Words 4 and 5 - Scatter/Gather List Element Address

These words contain the address of the scatter/gather list element being used when the command was ended.

Word 6 - Device Dependent Status Length

This word contains the number of bytes of device status information that follow.
This word is set to hex 0C to indicate 12 bytes.

Word 7 - Command and Device Status

The following table describes command status codes.

Hex	Command Status
1	SCB Command Completed with Success
5	SCB Command Completed with Success after Retries
7	Adapter Hardware Failure
A	Immediate Command Completed
C	Command Completed with Failure
E	Command Error (Invalid Command or Parameter)
F	Software Sequencing Error

Note: All values not shown are reserved.

The following table describes device status bytes.

Bit	Function
7	Reserved
6	Vendor Unique Bit
5	Vendor Unique Bit
4 - 1	Device Status Code
0	Vendor Unique Bit

The following table describes bytes 4-1 device status code.

Hex	Device Status
0	Good Status (No Error)
1	Check Condition (Error)
2	Condition Met/Good (No Error)
4	Busy (Error)
8	Intermediate/Good (No Error)
A	Intermediate/Condition Met/Good (No Error)
C	Reservation Conflict (Error)

Note: All values not shown are reserved.

Word 8 - Command Error Code/Device Error Code

The following table describes bits 15-8 command error codes.

Hex	Error
00	No Error
01	Invalid Parameter in SCB
02	Reserved
03	Command Not Supported
04	Command Aborted (By System)
05	Reserved
06	Reserved
07	Format Rejected - Sequence Error
08	Assign Rejected - Command in Progress on Device
09	Assign Rejected - SCSI Device Already Assigned
0A	Command Rejected - SCSI Device Not Assigned
0B	Maximum Logical Block Address Exceeded
0C	16-Bit Card Slot Address Range Exceeded.
0D - 12	Reserved
13	Invalid Device for Command
14 - 1F	Reserved
20	Adapter Hardware Error
21	Global Command Time-out

22	DMA Error
23	Adapter Buffer Defective
24	Command Aborted by Adapter
25 - 7F	Reserved
80	Adapter Microprocessor Detected Error
81 - FF	Reserved

The following table describes bits 7-0 device error codes.

Hex	Error
00	No Error
01	SCSI Bus Reset Occurred
02	SCSI Interface Fault
03 - 0F	Reserved
10	SCSI Selection Time-out (device not available)
11	Unexpected SCSI Bus Free
12	Reserved
13	Invalid SCSI Phase Sequence
14 - 1F	Reserved
20	Short Length Record
21 - FF	Reserved

Word 9 - Reserved

Word A - Cache Information Word

Bits 7 - 0 are the cache-read hit ratio (expressed as a percentage in a binary coded decimal format).

The following table describes casche-read hit rations.

Hex	Percent
00 - 99	00% - 99%
A0	100%

The following table describes bits 15-8 cache statuses.

Bit	Function
15 - 12	Reserved
11	Cache Enabled
10	Cache Retry Occurred
9	Total Write Hit
8	Total Read Hit

Word B - Last SCB Address Processed - Low Word

Word C - Last SCB Address Processed - High Word

Read Data

This SCB command is used for devices with fixed length blocks, such as fixed disk drives. This command causes the adapter to send the SCSI Read command to the device. The blocks specified are read and the data is transferred to the system.

The Read Data command supports multiple block operations up to 65,535 blocks or 16MB minus 1 byte (MB = 1,048,576 bytes), whichever is less, of total data transferred.

For devices with variable length blocks, such as tape drives, the Send Other SCSI SCB command should be used to generate the SCSI Read command.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	ND	NS	0	0	0	0	0	1	Read Data
1	ES	RE	PT	0	0	BB	0	0	0	0	0	0	0	0	CH	Enable Word
<-----				Least Significant Word				----->								Logical
<-----				Most Significant Word				----->								Address
<-----				Least Significant Word				----->								System Buffer
<-----				Most Significant Word				----->								Address
<-----				Least Significant Word				----->								System Buffer
<-----				Most Significant Word				----->								Byte Count
<-----				Least Significant Word				----->								Termination Status Block
<-----				Most Significant Word				----->								Address
<-----				Least Significant Word				----->								Optional SCB Chain
<-----				Most Significant Word				----->								Address
<-----				Number of Blocks				----->								Block Count
<-----				Block Size				----->								Block Length

Read Device Capacity

This SCB command is used to return the Device Capacity status block of the specific device.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	ND	NS	0	0	1	0	0	1	Read Device Capacity
1	ES	RE	0	0	0	1	0	0	0	0	0	0	0	0	CH	Enable Word
<-----			Reserved			<----->										
<-----			Reserved			<----->										
<-----			Least Significant Word										<----->		System Buffer	
<-----			Most Significant Word										<----->		Address	
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	System Buffer
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Byte Count
<-----			Least Significant Word										<----->		Termination Status Block	
<-----			Most Significant Word										<----->		Address	
<-----			Least Significant Word										<----->		Optional SCB Chain	
<-----			Most Significant Word										<----->		Address	
<-----			Reserved										<----->			
<-----			Reserved										<----->			

Device Capacity Data Block

Byte	7	6	5	4	3	2	1	0	Remarks
0	<----- High Byte ----->								Last Logical Block Address
1	<----->								
2	<----->								
3	<----- Low Byte ----->								
4	<----- High Byte ----->								Block Length
5	<----->								
6	<----->								
7	<----- Low Byte ----->								

Read Prefetch

For this SCB command, the blocks specified are read and the data is transferred into the on-card disk cache for later access by a Read Data command. The block length specified must be 512 bytes and the block count must be less than or equal to 17 for the command to transfer data into the cache. If other values are specified, the command is treated as a no-operation. This command is supported only by the cached adapter. The non-cached adapter will reject this command with a Command Error Interrupt ID. The presence of a cached adapter can be determined by bit 11 of the Cache Information word in the Command Complete Status Block. If this bit is set to 1, the adapter has a cache. Otherwise, no cache is present.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	ND	NS	1	1	0	0	0	1	Read Prefetch
1	ES	RE	0	0	0	1	0	0	0	0	0	0	0	0	CH	Enable Word
<-----			Least Significant Word ----->										Logical Block			
<-----			Most Significant Word ----->										Address			
<-----			Reserved ----->													
<-----			Reserved ----->													
<-----			Reserved ----->													
<-----			Reserved ----->													
<-----			Least Significant Word ----->										Termination Status Block			
<-----			Most Significant Word ----->										Address			
<-----			Least Significant Word ----->										Optional SCB Chain			
<-----			Most Significant Word ----->										Address			
<-----			Number of Blocks ----->										Block Count			
<-----			Block Size ----->										Block Length			

Read Verify

This SCB command reads the specified blocks of data and checks for errors. Data is not transferred by this command; it serves to verify the readability of the data and the correct operation of the device. This command is used for devices with fixed length blocks, such as fixed disk drives. This command causes the adapter to send the SCSI Read and Verify commands to the device. The blocks specified are read and the data is transferred to the system.

The Read Verify command supports multiple block operations up to 65,535 blocks or 16MB minus 1 byte (MB = 1,048,576 bytes), whichever is less, of total data transferred.

For devices with variable length blocks, such as tape drives, the Send Other SCSI SCB command should be used to generate the SCSI Read and Verify commands.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	ND	NS	0	0	0	0	1	1	Read Verify
1	ES	RE	0	0	0	1	0	0	0	0	0	0	0	0	CH	Enable Word
<-----			Least Significant Word ----->										Logical Block			
<-----			Most Significant Word ----->										Address			
<-----			Reserved ----->													
<-----			Reserved ----->													
<-----			Reserved ----->													
<-----			Reserved ----->													
<-----			Least Significant Word ----->										Termination Status Block			
<-----			Most Significant Word ----->										Address			
<-----			Least Significant Word ----->										Optional SCB Chain			
<-----			Most Significant Word ----->										Address			
<-----			Number of Blocks ----->										Block Count			
<-----			Block Size ----->										Block Length			

Reassign Block

This SCB command reassigns the logical block address for a defective block to a spare block. The system supplies the reassign block defect list. The system buffer address in the command block serves as a pointer to the defect list. Because the device data is considered erased by a Reassign Block command, the cache automatically clears any data from the device having a block reassigned.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	0	0	0	1	1	0	0	0	Reassign Block
0	ES	RE	0	0	0	1	0	0	0	0	0	0	0	0	CH	Enable Word
<----- Reserved ----->																
<----- Reserved ----->																
<----- Least Significant Word ----->																System Buffer
<----- Most Significant Word ----->																Address
<----- Least Significant Word ----->																System Buffer
<----- Most Significant Word ----->																Byte Count
<----- Least Significant Word ----->																Termination Status Block
<----- Most Significant Word ----->																Address
<----- Least Significant Word ----->																Optional SCB Chain
<----- Most Significant Word ----->																Address
<----- Reserved ----->																
<----- Reserved ----->																

Reassign Block Defect List

Byte	Defect List Header							Remarks
	7	6	5	4	3	2	1	
0	<----- Reserved ----->							
1	<----- Reserved ----->							
2	<----- High Byte ----->							Defect List Length
3	<----- Low Byte ----->							
	Defect Descriptors							
4	<----- High Byte ----->							Defective Logical Block Address
5	<----->							
6	<----->							
7	<----- Low Byte ----->							

Request Sense

This SCB command is used to return the sense data for the specified device. The adapter interrupts the system when the Sense data block is transferred. The length of the data block depends on the device and can be four bytes (non-extended) or more (extended). The format of the data block for both cases is shown. The system should specify the amount of data to be returned in the SCB based on the particular device attached, or specify the maximum value (255) and set the suppress short exception (SS) bit to 1.

The sense data is valid only if a Check Condition status was returned for the previous command to the device. The sense data provides additional information on the check condition. Refer to the ANSI SCSI publication or the particular SCSI device specification for detailed information about the Request Sense data block.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks	
0	0	0	1	1	1	0	0	ND	NS	0	0	1	0	0	0	Request Sense	
1	ES	RE	0	0	SS	1	0	0	0	0	0	0	0	0	CH	Enable Word	
<-----			Reserved			----->											
<-----			Reserved			----->											
<-----			Least Significant Word											----->		System Buffer	
<-----			Most Significant Word											----->		Address	
<-----			Least Significant Word											----->		System Buffer	
<-----			Most Significant Word											----->		Byte Count	
<-----			Least Significant Word											----->		Termination Status Block	
<-----			Most Significant Word											----->		Address	
<-----			Least Significant Word											----->		Optional SCB Chain	
<-----			Most Significant Word											----->		Address	
<-----			Reserved			----->											
<-----			Reserved			----->											

Sense Data Block

Byte	Sense Bits								Remarks
	7	6	5	4	3	2	1	0	
0	AV <Class> <- Code ->								Error Class/Code
1	X	X	X	< High Byte >					Logical
2	<----->								Block
3	<----- Low Byte ----->								Address

Byte 0 Error Class/Error Code

Bit 7	Address Valid: When this bit is set to 1, the logical block address field is valid.
Bits 6 - 4	Error Class: When the error class is 0, the sense data block is in the format shown above. When the error class is 7, the sense data block is in the extended format, shown on the following page. All other settings are device dependent.
Bits 3 - 0	Error Code: Errors are device dependent.

Bytes 1 - 3 Logical Block Address

This address is device dependent.

Note: The adapter does not examine or use device-dependent information.

Extended Sense Data Block

Byte	Sense Bits								Remarks
	7	6	5	4	3	2	1	0	
0	V	1	1	1	<-- Code -->				Error Class/Code
1	<----- Segment # ----->								Segment Number
2	FM	EM	IL	X <-- Key -->					Sense Key
3	<- Most Significant -->								Information Bytes
4	<----->								
5	<----->								
6	<- Least Significant ->								
7	<--Additional Length-->								# of Bytes
8	<--Additional Sense -->								Additional
	.								Sense
	.								
N	<--Additional Sense--->								Bytes

Byte 0 Error Class/Error Code

Bit 7	The Information bytes are valid only if this bit is a 1.
Bits 6 - 4	Error class 7 is for extended sense data.
Bits 3 - 0	Error code 0 is standard format. Error codes hex 1 - E are reserved. Error code hex F is device dependent.

Byte 1 Segment Number

This byte contains the current segment descriptor.

Byte 2 Extended Error Bits/Sense Key

Bit 7	A filemark (FM) has been reached on a sequential access device.
Bit 6	An end of medium (EM) has been reached on a sequential access device.
Bit 5	An Invalid Length (IL) resulted when the specified logical block length did not match the device.
Bit 4	X - This bit is reserved.
Bits 3 - 0	The coding of these bits is shown in the following table.

Hex Value	Function
0	No Sense
1	Recovered Error
2	Not Ready
3	Medium Error
4	Hardware Error
5	Illegal Request
6	Unit Attention
7	Data Protect
8	Blank Check
9	Device Dependent
A	Copy Aborted
B	Aborted Command
C	Equal

D	Volume Overflow
E	Miscompare
F	Reserved

Bytes 3 - N **Device-Dependent Status:** Refer to the particular device specifications for a definition of these bytes.

Note: The adapter does not examine or use device-dependent information.

Send Other SCSI Command (SCSI CDB)

This SCB command is used to send any SCSI command not supported by the adapter directly to a SCSI device. The command to be issued is placed at the end of the SCB. When commands are issued directly to a device using this command, messages are handled by the adapter. Data transfer direction is controlled by the read-option bit (RD) in the Enable word. When this bit is set to 1, the adapter transfers data to the system from the device. When the read-option bit is set to 0, the adapter transfers data to the device from the system. If the system-buffer byte count specified in the SCB is 0, no data is transferred. Because device data can be altered by this command, the cache is automatically cleared of any data from that device.

Note:

This command should be used only when other commands cannot perform the operation; otherwise, performance of the SCSI subsystem can be impacted.

This command should be issued only to logical device numbers 0 to 14.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	1	0	0	1	0	0	ND	NS	0	1	1	1	1	1	Send Other SCSI Command
RD	ES	RE	PT	0	SS	1	0	0	0	0	0	0	0	0	CH	Enable Word
<----- Reserved -----><-- SCSI CMD Length --->																
<----- Reserved ----->																
<----- Least Significant Word ----->																System Buffer
<----- Most Significant Word ----->																Address
<----- Least Significant Word ----->																System Buffer
<----- Most Significant Word ----->																Byte Count
<----- Least Significant Word ----->																Termination Status Block
<----- Most Significant Word ----->																Address
<----- Least Significant Word ----->																Optional SCB Chain
<----- Most Significant Word ----->																Address
<-----1----- SCSI Command -----0----->																SCSI Command
<-----3----- SCSI Command -----2----->																
<-----5----- SCSI Command -----4----->																Six Bytes or
<-----7----- SCSI Command -----6----->																
<-----9----- SCSI Command -----8----->																Ten Bytes or
<-----11----- SCSI Command -----10----->																Twelve Bytes

Write Data

This SCB command writes data from the system to the device in consecutive blocks. This command is used for devices with fixed length blocks, such as fixed disk drives. This command causes the adapter to send the SCSI Write command to the device. The blocks specified are read and the data is transferred to the system. No verification is performed.

The Read Data command supports multiple block operations up to 65,535 blocks or 16MB minus 1 byte (MB = 1,048,576 bytes), whichever is less, of total data transferred.

For devices with variable length blocks, such as tape drives, the Send Other SCSI SCB command should be used to generate the SCSI Write command.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	ND	NS	0	0	0	0	1	0	Write Data
0	ES	RE	PT	0	0	BB	0	0	0	0	0	0	0	0	CH	Enable Word
<-----				Least Significant Word				----->				Logical Block				
<-----				Most Significant Word				----->				Address				
<-----				Least Significant Word				----->				System Buffer				
<-----				Most Significant Word				----->				Address				
<-----				Least Significant Word				----->				System Buffer				
<-----				Most Significant Word				----->				Byte Count				
<-----				Least Significant Word				----->				Termination Status Block				
<-----				Most Significant Word				----->				Address				
<-----				Least Significant Word				----->				Optional SCB Chain				
<-----				Most Significant Word				----->				Address				
<-----				Number of Blocks				----->				Block Count				
<-----				Block Size				----->				Block Length				

Write with Verify

This SCB command is similar to Write Data, except that a Read Verify command is performed after all blocks are written. This command is used for devices with fixed length blocks, such as fixed disk drives. This command causes the adapter to send the SCSI Write and Verify commands to the device. The blocks specified are read and the data is transferred to the system.

The Write with Verify command supports multiple block operations up to 65 535 blocks or 16MB minus 1 byte (MB = 1,048,576 bytes), whichever is less, of total data transferred.

If an error occurs during a Write with Verify command, the system should retry the command. If all retries of the command fail, the system can allocate a spare block to replace the failing one through the Reassign Block command, and then reissue the command.

For devices with variable length blocks, such as tape drives, the Send Other SCSI SCB command should be used to generate the SCSI Write and Verify commands.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Remarks
0	0	0	1	1	1	0	0	ND	NS	0	0	0	1	0	0	Write Verify
0	ES	RE	PT	0	0	BB	0	0	0	0	0	0	0	0	CH	Enable Word
<-----				Least Significant Word				----->								Logical Block
<-----				Most Significant Word				----->								Address
<-----				Least Significant Word				----->								System Buffer
<-----				Most Significant Word				----->								Address
<-----				Least Significant Word				----->								System Buffer
<-----				Most Significant Word				----->								Byte Count
<-----				Least Significant Word				----->								Termination Status Block
<-----				Most Significant Word				----->								Address
<-----				Least Significant Word				----->								Optional SCB Chain
<-----				Most Significant Word				----->								Address
<-----				Number of Blocks				----->								Block Count
<-----				Block Size				----->								Block Length

Advanced SCSI Programming Interface (ASPI) OS/2 Specification

The programming specification for interfacing with the OS2ASPI.DMD device manager may be obtained by contacting the Adaptec Corporation in Milpitas, California, phone number (800) 934-2766 5AM-6PM PST.

Adapter Device Driver Interface Questions and Answers

This appendix covers the most commonly asked questions about adapter device driver interfaces, including discussion of scatter/gather lists and functionality. The answers are presented in detail.

Scatter/Gather Lists

Question:

Is there a limit on the length of an individual scatter/gather element?

Answer:

No. We are aware that some SCSI PIO drivers have a 64KB limitation on scatter/gather elements. At present, OS/2 2.0 appears to remain within this limitation. However, it is recommended that this restriction be lifted because there is no guarantee that adapter device drivers with such a limitation will be compatible with future OS/2 operating system releases.

Question:

Is there a limit on the number of elements in a scatter/gather list?

Answer:

No.

Question:

What is the meaning of the field:

AdapterInfo->UnitInfo->MaxHWSGList

Answer:

This is the maximum number of scatter/gather list entries your adapter hardware can handle. The OS/2 operating system will ensure that the adapter device driver is not required to split a sector across a MaxHWSGList boundary. However, the adapter device driver is responsible for iterating an I/O command because of limitations in its s/g support.

Question:

What about s/g lists associated with CDB PassThru commands?

Answer:

adapter device drivers are not required to iterate commands through the CDB PassThru mechanism.

Commands received from SCSI.SYS device class drivers will require a minimum of 16 s/g elements. At present, the number of s/g elements required by device modules written to ASPI interfaces is not known. If the OS2SCSI or OS2ASPI device managers receive an s/g list that does not conform to the adapter device driver's s/g requirement, these device managers will reject the request.

Question:

What is the meaning of the field:

AdapterInfo->UnitInfo->MaxCDBXferLen

Answer:

This field is *not* the maximum CDB length. The purpose of this field relates to the limitations on s/g list lengths and their effect on PassThru commands.

For adapter device drivers written for adapters that have severe s/g list limitations or unusual s/g address limitations, the adapter device driver might emulate s/g functionality using an I/O buffer. In this case, the adapter device driver would inform the OS2SCSI or OS2ASPI device managers of the length of its emulation buffer by way of the above field. It is up to the adapter device driver to perform this emulation.

Question:

My adapter requires an s/g list format different from the one provided. Do I need to allocate storage for separate lists?

Answer :

No. Provided that your transformations are reversible, you can reformat the passed s/g list to what your adapter can accept. You must reverse the reformatting prior to performing your notification call out.

Adapter Device Driver Functionality

Question:

Do I need to support every command in this specification?

Answer:

No. For specific device types, some commands are mandatory and others are optional. For example, there are several commands that apply only to floppy controllers. We must support these commands due to the unusual characteristics of diskette media.

Where possible, mandatory commands for particular device types are indicated.

Question:

Some commands do not appear to be used by the current OS/2 2.0 device managers. Must I implement these commands?

Answer:

Yes. To preclude compatibility problems with future releases of OS/2, you should follow this specification as closely as possible with respect to mandatory commands.

Question:

What happens if IBM-supplied sample code differs in some way from this specification?

Answer:

This specification takes precedence over IBM-supplied samples. Obtain clarification from IBM if you have questions on a particular code sample.

Question:

SCSI defines different command formats for different device types. Do I need to support all the different CDB formats?

Answer:

No. You must support the DASD type formats only. The remainder of the device types will be supported by way of CDB PassThru; that is, the adapter device driver will be supplied with an appropriate CDB.

Question:

Do I need to support the IORB_CHAIN *RequestControl* flag on every IORB command type?

Answer:

No. The chain flag is used only on IOCC_EXECUTE_IO commands. In addition, all IORBs contained in the chained request will reference the same *UnitHandle*.

Question:

Do I need to support any unusual SCSI commands?

Answer:

No. We expect most DASD devices to conform to CCS. Regarding SCSI Write/Verify, adapter device driver writers are encouraged to emulate this command for drives that do not provide this function directly. Suppliers of devices that have unusual command requirements are expected to provide filter device drivers.

Question:

What kind of error recovery is an adapter device driver required to perform?

Answer:

Adapter device drivers are responsible for error recovery. An error reported to the layers above the adapter device driver is considered not recoverable. In general, SCSI devices retry at the device level, so the adapter device driver does not need to retry commands that failed at the device. However, the adapter device driver should retry commands that failed due to errors on the SCSI bus during either the command or data transfer phases.

Question:

Must the adapter device driver retry commands that were disrupted due to an adapter reset for a *hang* condition?

Answer:

Yes, the adapter device driver is responsible for retrying commands disrupted by an adapter device driver-initiated reset of an adapter to clear a hang condition.

Question:

Does the adapter device driver have to report media removal with an error code to the device manager?

Answer:

If the unit supports media removal, the adapter device driver is required to report media removal to the device manager by way of the appropriate IOERR_* code. In addition, the adapter device driver should return the same error code for all work queued for the unit. The adapter device driver should not retry this error. The adapter device driver should treat a power-on condition the same as media removal. The adapter device driver should return the appropriate IOERR_* code for all queued work after a not ready condition.

If the unit does not support media removal, the adapter device driver should retry the operation.

Question:

Must my adapter device driver perform SCSI sense code->IOERR_* translation?

Answer:

Yes. The OS/2 DASD device manager expects a uniform set of error codes and will not examine SCSI sense codes.

Question:

Must my adapter device driver return sense data when requested?

Answer:

Yes. The OS2SCSI and OS2ASPI device managers need to return this data to their

clients.

Notices

The following terms, denoted by an asterisk ((*)) in this publication, are trademarks of the IBM Corporation in the United States or other countries:

IBM	MCA
IBM Personal System/2	Micro Channel
IBM PS/2	OS/2
	Presentation Manager

The following terms, denoted by a double asterisk (**)) in this publication, are trademarks of other companies as follows:

Adaptec	Adaptec Corporation
Xenix	Microsoft Corporation

Glossary

This glossary contains terms and definitions that are, for the most part, used for OS/2 products. This is not a complete dictionary of computer terms.

Introduction

This glossary defines many of the the terms used in this book. It includes terms and definitions from the *IBM Dictionary of Computing*, as well as terms specific to the Presentation Manager, but it is not a complete glossary for OS/2.

Other primary sources for these definitions are:

- o The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyrighted 1990 by the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. These definitions are identified by the symbol (A) after the definition.
- o The *Information Technology Vocabulary*, developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

Glossary

Select a starting letter of glossary terms:

A	N
B	O
C	P
D	Q
E	R
F	S
G	T
H	U
I	V
J	W
K	X
L	Y
M	Z

Glossary - A

A

ABIOS - Advanced BIOS. See *BIOS*.

accelerator - A single keystroke that invokes an application-defined function.

accelerator table - A table that defines which keystrokes are treated as *accelerators*, and the commands into which they are translated.

access permission - All access rights a user has regarding an object. (I)

action - One of a set of defined tasks performed by a computer. A user requests the application to perform an action in several ways, such as typing a command, pressing a function key, or selecting the action's name from an action bar or menu.

action bar - The area at the top of a window that contains the choices that are currently available in the application program. When the user selects an action bar choice, a group of actions or additional keywords appear in a pull-down extension from the action bar.

action point - The current position on the screen at which the pointer is pointing. Contrast with *hotspot* and *input focus*.

active program - A program currently running on the computer. An active program can be "interactive" (running and receiving input from the user), or "noninteractive" (running, but not receiving input from the user). See also *interactive program*, *noninteractive program*, and *foreground program*.

active window - In *Common User Access** architecture, the window with which the user is currently interacting. This is the window that receives keyboard input. Contrast with *inactive window*.

adapter - A piece of hardware that modifies the system unit to allow it to operate in a particular way, often by connecting the system unit to an external device such as a video monitor.

adapter device driver - A device driver that provides hardware-dependent services for an *OEM* adapter.

address bus - The path used for the transmission of address information in a computer.

address space - (1) The range of addresses available to a program. (A)

(2) The area of virtual storage available for a particular job.

All-Points-Addressable (APA) - Pertaining to the ability of a terminal device such as a printer or a display monitor to address and display (or not display), each picture element (pel) on a display surface.

alphanumeric video output - Output to the logical video buffer when the video adapter is in text mode, and the logical video buffer is addressed by an application as a rectangular array of character cells.

American National Standard Code for Information Interchange (ASCII) - The standard code, using a coded character set consisting of seven-bit coded characters (eight bits including parity check), used for information interchange among data processing systems, data communications systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

Note: IBM has defined an extension to ASCII code (characters 128-255).

American National Standards Institute (ANSI) - An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

anchor block - An area of the internal resources of OS/2 Presentation Manager* which is allocated to a process or thread that calls [WinInitialize](#).

anchor point - A point in a window used by a program designer or by a window manager to position a subsequent window.

ANSI - American National Standards Institute

APA - All-Points-Addressable.

API - Application Programming Interface.

Application Programming Interface (API) - A functional interface supplied by the operating system, or by a separately-orderable licensed program, that allows an application program written in a high-level language to use specific data or functions of the operating system or the licensed program.

See also *Graphics Programming Interface (GPI)*.

area - In computer graphics, a filled shape such as a solid rectangle.

ASCII - American National Standard Code for Information Interchange.

ASCIIIZ - A string of ASCII characters that is terminated with a byte containing a null character.

aspect ratio - The width-to-height ratio of an area, symbol, or shape.

asynchronous (ASYNCH) - (1) Without regular time relationship.

(2) Unexpected or unpredictable with respect to the execution of program instructions.

Contrast with *synchronous*.

atom - A constant that represents a string. Once a string has been defined as an atom, the atom can be used in place of the string to save space. Strings are associated with their respective atoms in an *atom table*. See *integer atom*.

atom table - A table used to associate *atoms* with the strings that they represent. This table contains the mechanism by which the presence of a string can be verified.

attributes - Characteristics or properties that can be controlled, usually to obtain a required appearance; for example the color or width of a line. Compare with *graphics attributes* and *segment attributes*.

AVIO - Advanced Video Input/Output

Glossary - B

B

background color - The color in which the background of a *graphic primitive* is drawn.

background mix - An attribute that determines how the background of a *graphic primitive* is combined with the existing color of the graphics presentation space. Compare with *mix*.

background program - In *multiprogramming*, a program that executes with a low priority. Contrast with *foreground program*.

base device driver - An OS/2 device driver that performs I/O during the OS/2 kernel boot sequence to provide IPL support. Base device drivers are loaded by way of the CONFIG.SYS **BASEDEV** keyword, rather than the **DEVICE** keyword. See *BASEDEV keyword*, *adapter device driver*, and *device manager*.

BASEDEV keyword - New CONFIG.SYS keyword; loads a base device driver into the operating system.

Basic Input/Output System (BIOS) - In an IBM personal computer, microcode that controls basic hardware operations such as interactions with diskette drives, fixed disk drives, and the keyboard.

Bezier curve - A mathematical technique of specifying a smooth, continuous line or surface, requiring a starting point and an ending point, with several intermediate points that influence or control the path of the linking curve. Also called a "spline" or a "B-spline".

BIOS - Basic Input/Output System.

bit-block (bitblt) function - Draws the rectangles that compose the OS/2 Presentation Manager Desktop, icons, and other *bit maps*.

bit map - A graphic representation of an image on an APA device (usually a monitor screen) by an array of binary digits.

block - (1) A string of data elements recorded or transmitted as a unit. The elements may be characters, words, or physical records.

(2) To combine two or more data elements in one block.

Block Logical Transfer (BLT) - The process of transferring one or more blocks of data.

BLT - Block Logical Transfer.

border - A visual indication of the boundaries of a window, for example, a separator line or a background color.

BPB - BIOS Parameter Block.

breakpoint - (1) An instruction in a program for halting execution. A breakpoint is usually at the beginning of an instruction where halts, caused by external intervention, are convenient for restarting. (T)

(2) A place in a program, specified by a command or condition, where the system halts execution and gives control to the workstation user or to a specified program.

bucket - One or more fields in which the result of an operation is kept.

buffer - (1) A portion of storage used to temporarily hold input or output data.

(2) To allocate and schedule the use of buffers. (A)

bus - One or more conductors used for transmitting signals or power. (A)

Bus Master - A device or subsystem that controls data transfers between itself and a slave.

Bus Master adapter - An adapter capable of performing Reads and Writes to physical storage by communicating directly with the storage subsystem (memory) rather than depending on a host DMA channel or host CPU. Synonymous with *first-party DMA adapter*.

button - A switch mechanism on a *pointing device* such as a mouse or a trackball, used to request or initiate an action. Compare with *push button* and *radio button*. See also *select button*.

Glossary - C

C

cache - A high-speed buffer storage that contains frequently-accessed instructions and data; it is used to reduce access time.

cached micro presentation space - A presentation space from a store of micro presentation spaces owned by the Presentation Manager. It can be used for drawing to a window only, and must be returned to the store when the task is complete.

call - (1) The action of bringing a computer program, routine, or subroutine into effect, usually by specifying the entry conditions and jumping to an *entry point*. (I) (A)

(2) To transfer control to a procedure, program, routine, or subroutine.

calling order - A sequence of instructions together with any associated data necessary to perform a call. Synonymous with *calling sequence*.

calling sequence - See *calling order*.

cancel - An action that removes the current window or menu without processing it, and returns the previous window.

CASE - Computer-Aided Software Engineering

CASE statement - In C/2, provides the body of a window procedure. There is one CASE statement for each message type written to take specific actions.

CCS - (1) Common Command Set.

(2) Common Communications Support.

CD-ROM - Compact Disc-Read Only Memory. High-capacity read-only memory in the form of an optically-read compact disc.

CDB - Command Descriptor Block.

cell - See *character cell*.

CGA - Color Graphics Adapter.

chained list - Synonymous with *linked list*.

chaining - A method of storing records in which each record belongs to a list or group of records and has a linking field for tracing the chain. See the illustration at *linked list*.

character - A single letter, digit, or other symbol.

character box - In computer graphics, the boundary that defines, in world coordinates, the horizontal and vertical space occupied by a single character from a character set. Contrast with *character cell*. See *character mode*.

character cell - The physical, rectangular space in which any single character is displayed on a screen or printer device. Position is addressed by row and column coordinates. Contrast with *character box*.

character code - The means of addressing a single character in a character set. Synonymous with *code point*.

character mode - The character mode, in conjunction with the font type, determines the extent to which graphics characters are affected by the character box, shear, and angle attributes.

check box - In *Common User Access* architecture, a *control window* in the form of small box on the screen, with associated text, that represents a choice. When the user selects a choice, an " X " appears in the check box to indicate that the *checked state* is in effect. The check box can be cleared (returned to the *unchecked state*) by selecting the choice again. Contrast with *radio button* and *push button*.

checked state - See *check box*.

check mark - The symbol (¹) that is used to indicate a selected item on a *pull-down*.

child process - A process that is loaded and started by another (parent) process and which shares the resources of that process. Contrast with *parent process*.

child window - A window that is positioned relative to another (either a main window or another child window). Contrast with *parent window*, compare with *sibling window*.

choice - In *Common User Access* architecture, an option that can be selected by a user. The choice can be presented as text, a symbol (number or letter), or an icon (a pictorial symbol).

class - See *window class*.

class style - The set of properties that apply to every window in a *window class*.

client area - The area in the center of a window that contains the primary information of the window.

clipboard - An area of main storage that can hold data being passed from one OS/2 Presentation Manager application to another. Various data formats can be stored.

clipping - In computer graphics, removing those parts of a display image that lie outside a given boundary.

clip limits - The area of the paper that can be reached by a printer or plotter.

clipping path - A clipping boundary in world-coordinate space.

CLOCK\$ - Character-device name reserved for the system clock.

code page - A set of assignments of graphic characters and control-function meanings to all *code points*.

code point - Synonymous with *character code*.

code segment - An executable section of programming code within a load module.

color conversion - Changing one color format to another. Required, for example, when the source color format is different from the destination color format.

When going from the monochrome color format to the color format, 1 (one) bits are converted to the image foreground color, and 0 (zero) bits are converted to the image background color.

When going from color to monochrome, all pels that match the passed background color are converted to the image background color of the destination.

All other pels are converted to the image foreground color of the destination. The color conversion takes place prior to any mix mode.

color dithering - See *dithering*.

Color Graphics Adapter (CGA) - An *adapter* of intermediate resolution, that provides four colors.

command - (1) The name and parameters associated with an action that a program can perform. A command is one form of action request.

(2) An action a user requests to interact with the command area.

command area - An area composed of a command field prompt and a *command entry point*.

command code - In this specification, refers to a group of related commands that an adapter device driver can receive.

All command codes have a prefix of "IOCC_". For example, common I/O requests (such as Read, Write, etc.) are grouped under the command code [IOCC_EXECUTE_IO](#).

command data block - A data structure defined by the *Small Computer System Interface* standard to send commands to devices that conform to SCSI standards.

command descriptor block (CDB) - The structure used to communicate commands from a source to a destination.

command entry point - An entry field into which the user types commands.

command line - On a display screen, a display line, usually at the bottom of the screen, in which only commands can be entered.

command modifier - In this specification, a specific operation that an adapter device driver is to perform.

All command modifiers have a prefix of "IOCM_". For example, an adapter device driver might receive an [IOCC_EXECUTE_IO](#) command with a command modifier of [IOCM_READ](#).

command prompt - A field prompt showing the location of the command entry field in a panel.

Common Communications Support (CCS) - Protocols and conventions for connecting systems and software.

One of the three *SAA* architectural areas. See *Systems Application Architecture*, *Common User Access*, and *Common Programming Interface*.

Common Programming Interface (CPI) - A consistent set of specifications for languages, commands, and calls to enable applications to be developed across the *SAA* environments.

One of the three *SAA* architectural areas. See *Systems Application Architecture*, *Common User Access*, and *Common Communications Support*.

Common User Access (CUA) architecture - A set of guidelines that define the way information is presented on the screen, and techniques for the user to interact with that information.

One of the three *SAA* architectural areas. See *Systems Application Architecture*, *Common Communication Support*, and *Common Programming Interface*.

compatibility kernel - The portion of the OS/2 kernel that exists to support DOS INT 20, 21, 25, 26, and 27 functions. It acts as an interface to common kernel functionality such as the file system.

compile - To translate a program written in a higher-level programming language into a machine-language program.

CON - Character-device name reserved for the CONsole keyboard and screen.

conditional compilation - Processing by the preprocessor of certain specified code in the file, depending on the evaluation of a specified condition.

contention - The state in which a DOS session attempts to access a serially-reusable resource (such as a COM device) while another DOS session or an OS/2 application is using the resource.

context hook - Similar to a "force flag" in earlier versions of OS/2. These are events, signaled by a virtual device driver, that are processed at task time. Forcing an IRET, and simulating an NMI, can fall into this category.

contiguous - Touching or joining at a common edge or boundary, for example, an unbroken consecutive series of storage locations.

contingent allegiance - A condition, typically generated by a **CHECK CONDITION** status, during which a destination preserves sense data.

control - (1) The means by which an operator gives input to an application.

(2) In *Common User Access* architecture, a component of the user interface that allows the user to select *choices* or to enter information by typing from the keyboard, as in a *check box*, *radio button*, or entry field.

control panel - In the Presentation Manager, a program used to set up user preferences that then act globally across the system.

control program - The basic function of OS/2, including DOS emulation and support for keyboard, pointing device, and video input/output.

control window - A class of window used to handle a specific kind of user interaction. *Radio buttons*, *push buttons*, and *check boxes* are examples of control windows.

controller sector buffer - One or more buffers, managed by a hardware adapter, to improve I/O transfer rates by helping to match a device and software timing requirements.

CPI - Common Programming Interface.

CRF - Client Register Frame

cursor - A symbol displayed on the screen and associated with an input device. The cursor indicates where input from the device will be placed.

Contrast with *pointer* and *input focus*.

Glossary - D

D

DAC - Digital-to-Analog Converter.

DASD - Direct-Access Storage Device.

DASD geometry - The organization of direct-access devices, including the cylinder/head/sector layout, RBA capacity, and block size of the device.

data stripping - A technology for spreading to multiple disks the information that would normally be written to only one disk.

data structure - The syntactical structure of symbolic expressions, and their storage allocation characteristics.

DBCS - Double-Byte Character Set.

DC - Device Context

DDB - Device-Dependent Bit map.

default value - A value supplied by the program and used when no other value is explicitly specified by the user. For example, in the GPI, the default line type is "solid".

deinstantiation - See *instantiation*.

Desktop Manager - In the Presentation Manager, a window from which a user can start one or more listed programs.

desktop window - The window corresponding to the physical device against which all other types of windows are established.

DevHlp - Device helper.

device context - A logical description of a data destination such as memory, metafile, display, printer, or plotter. See also *direct device context*, *information device context*, *memory device context*, *metafile device context*, and *screen device context*.

device driver - (1) A program that enables the computer to communicate with a specific peripheral device, for example, a printer, a video disc player, or a CD-ROM drive.

(2) A code module written to the specifications of the OS/2 device drivers.

device driver initialization (init) time - See *initialization (init) time, device driver*.

device driver profile - A file with a "DDP" extension, containing a script that is interpreted by the OS/2 DDINSTALL utility. Among other things, it defines which files to copy from installation diskettes to target directories and specifies how the CONFIG.SYS file will be updated.

device helper (DevHlp) - (1) A kernel service (memory, hardware interrupt, software interrupt, queuing, semaphore, and so forth) provided to physical device drivers.

(2) A callable C-language or assembler-language routine that provides an operating system service for an OS/2 device driver.

Device Manager (DM) - An OS/2 device driver that interfaces with other OS/2 device drivers, OS/2 installable file systems, or the OS/2 kernel, and converts requests to conform to this specification.

device object - A device that provides a means of communication between a computer and the outside world. A printer is an example of a device object.

device table - A data structure containing a summary of the adapters an adapter device driver supports and a list of the I/O devices attached to each adapter. This data structure is built by the adapter device driver in response to an [IOCC_CONFIGURATION IOCM_GET_DEVICE_TABLE](#) request.

dialog - The interchange of information between a computer and its user through a sequence of requests by the user and the presentation of responses by the computer.

dialog box - A type of window that contains one or more controls for the formatted display and entry of data. Also called a *pop-up window*. A *modal dialog box* is used to implement a pop-up window.

dialog box editor - A *WYSIWYG* editor that is used to create dialog boxes for communication with the user.

dialog item - A component (such as a menu or a push button) of a dialog box. Dialog items are also used to create templates.

dialog template - The definition of a dialog box, containing details of its placement, appearance, window ID, and the window IDs of all its *child windows*.

DIB - Device-Independent Bit map

Direct-Access Storage Device (DASD) - A device in which data-access time is effectively independent of the location of the data in the storage medium.

direct manipulation - The use of the *pointing device* to move objects around on the screen, as in moving files and directories about in the File Manager.

Direct Memory Access (DMA) - (1) A technique for moving data directly between main storage and peripheral equipment without requiring processing of the data by the processing unit.

(2) The transfer of data between memory and input/output units without processor intervention.

directory - A type of file containing the names and controlling information for other files and (sub)directories.

Disk Operating System (DOS) - An operating system for computer systems that uses disks and diskettes for auxiliary storage of programs and data.

dispatch table - (1) A block of memory, allocated by the graphics engine, for the containment of entry points for use by a display driver.

(2) An array of pointers to function-handling routines.

display point - Synonymous with *pel*.

dithering - A technique for interleaving dark and light pels so that the resulting image looks smoothly shaded from a distance.

DLL - Dynamic-Link Library.

DMA - Direct Memory Access.

DMA slave - An adapter that requires a host DMA channel to perform data transfers on its behalf. Also referred to as a *second-party DMA adapter*.

DOS - Disk Operating System.

DOS session - A session created by the operating system, that supports the independent processing of a DOS program.

DOS session breakpoint - (1) A mechanism to regain control from a DOS session; a V86 IRET frame is edited to point to an illegal V86 instruction, and the original CS:IP is saved in a table indexed by the address of the illegal instruction.

(2) A byte of memory visible in each DOS session containing an illegal instruction.

Double-Byte Character Set (DBCS) - A set of characters in which each character is represented by two bytes. Languages such as Japanese, Chinese, and Korean, which contain more characters than can be represented by 256 code points, require double-byte character sets. Since each character requires two bytes, the entering, displaying, and printing of DBCS characters requires hardware and software that can support DBCS.

DPMI - DOS Protect-Mode Interface.

drag - To move an object on the display screen as if the object were attached to

the cursor.

driver - A program and possibly data files that contain information needed to run a particular unit, for example, a plotter, printer, port, or mouse.

drop - To release an object that was being *dragged*.

DSC - DiSplay Configuration.

DSP - DiSPlay.

DTL - Dialog Tag Language.

DTT - Display Test Tool.

Dynamic Link Library (DLL) - A module containing a *Dynamic Link Routine* that is linked at load time or run time.

Dynamic Link Routine (DLR) - A program or routine that can be loaded by a program or as part of an application.

Glossary - E

E

EBCDIC - Extended Binary-Coded Decimal Interchange Code.

EGA - Enhanced Graphics Adapter.

encryption - The process of transforming data into an unintelligible or unreadable form in such a way that the original data can be obtained only by using a decryption process.

entry field - A panel field into which a user can enter information. Compare with *selection field*.

entry-field control - The means by which the application receives data which was entered by the user into an entry field. When it has the *input focus* it displays a flashing cursor at the position where the next typed character will appear.

entry panel - A defined panel type containing one or more *entry fields* together with protected information such as headings, prompts, and explanatory text.

entry point - A programming interface that refers to the point in code where the thread of execution is going to enter that routine.

environment - The computing context presently in use.

EOI - End Of Interrupt

.EXE - EXEecute. A file name extension for an *object* that starts a program.

exit - The action that terminates the current function and returns the user to the next higher-level function. Repeated exit requests eventually return the user to the point from which all functions are accessible. Contrast with *cancel*.

extended-choice selection - A mode that allows the user to select more than one item from a window. Not all windows allow extended choice selection. Contrast with *multiple-choice selection*.

extended help - A facility that provides the user with information about an entire application panel rather than only on a particular item on the panel.

Glossary - F

F

Far Call - Code that calls from one segment into another segment.

field-level help - Information specific to the field on which the cursor is positioned. This help function is "contextual" because it provides information about a specific item as it is currently used. The information is dependent on the context within the work session.

FIFO - First-In-First-Out.

File Manager - In the Presentation Manager, a program that displays directories and files and allows various actions on them.

file specification - The full identifier for a file which includes its file name, extension, path and drive.

fillet - An arc that is tangential to the end points of two adjacent lines. See also *polyfillet*.

filter adapter device driver - A special class of adapter device drivers that do not manage the hardware directly, but monitor the stream of commands between a device manager and an adapter device driver. See *Device Manager* and *adapter device driver*.

first-party DMA adapter - See *bus master adapter*.

First-In-First-Out (FIFO) - A data queueing technique in which the next item to be retrieved is the item that has been in the queue for the longest time.

flag - (1) A variable indicating that a certain condition holds. (T)

(2) A character that signals the occurrence of some condition, such as the end of a word. (A)

flat address - See *linear address*.

font - A particular size and style of typeface that contains definitions of character sets, marker sets and pattern sets.

foreground program - (1) The program with which the user is currently interacting.

(2) In *multiprogramming*, a program that executes with a high priority.

Also called an *interactive program*.

frame - The part of a window that can contain several different visual elements specified by the application but drawn and controlled by the Presentation Manager. The frame encloses the client area.

frame styles - Different standard window layouts provided by the Presentation Manager.

freeze and thaw services - Functions that prevent a DOS session from executing ([VDHFreezeVDM](#)) until the matching thaw function ([VDHThawVDM](#)) is called. The freeze occurs when the specified DOS session leaves *kernel mode*.

full-screen application - An application program that occupies the whole screen.

function - (1) In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a *call*.

(2) A set of related control statements that cause one or more programs to be performed.

function key - A key that causes a specified sequence of operations to be performed when it is pressed; for example, F1.

Glossary - G

G

GDC - Graphics Data Controller.

GDT - Global Descriptor Table.

Global Descriptor Table (GDT) - A table that defines code and data segments available to all tasks in an application.

glyph - A graphic symbol whose appearance conveys information.

GPI - Graphics Programming Interface

Graphical User Interface (GUI) - A type of user interface that takes advantage of high-resolution graphics. In common usage, a GUI includes a combination of graphics, object-action paradigms, pointing devices, menu bars and other menus, overlapping windows, and icons.

graphic primitive - In computer graphics, a basic element, such as an arc or a line, that is not made up of smaller parts and that is used to create diagrams and pictures.

graphics - An image defined in terms of *graphic primitives* and *graphics attributes*.

graphics attributes - The attributes that apply to graphics primitives. Examples are color selection, line type, and shading pattern definition.

Contrast with *segment attributes*.

Graphics Programming Interface (GPI) - The formally-defined programming language that lies between an IBM graphics program and the user of the program. See also *API*.

graphics segment - (1) In GDDM, a group of graphics primitives (lines, arcs, and text) that are operated as a common set.

(2) The graphics primitives inside a graphics segment share characteristics, such as visibility and angle of rotation, but keep their individual characteristics, such as color, line, and width.

GRE - Graphics Engine.

GUI - Graphical User Interface.

graying - The indication that a choice on a pull-down is unavailable.

group - A collection of logically-connected controls. For example, the buttons controlling paper size for a printer. See also *program group*.

GUI - Graphical User Interface.

Glossary - H

H

handle - An identifier that represents an object, such as a device or a window.

handshaking - A method by which two pieces of hardware, such as a personal computer and a plotter, can communicate. Depending on the devices communicating, handshaking occurs either as a hardware function or through software, such as a device driver.

hardcopy - Physical output (such as paper, slides or transparencies) from an output device, most usually a printer or a plotter.

hard error - An error condition that requires either that the system be reconfigured or that the source of the error be removed before the system can resume reliable operation.

hardware palette - The array of RGBs that the physical device is displaying.

heartbeat processing - A mechanism to increment and check a counter (the heartbeat) to determine if a seamless windows DOS session has failed while holding the video hardware semaphore.

heap - An area of free storage available for dynamic allocation by the program. Its size varies depending on the storage requirements of the program.

help - A function that provides information about a specific field, an application panel or information about the help facility. It provides field help when the cursor is on a selection or entry field in an application panel or another help panel. It provides information about the application panel, called *extended help*, when the cursor is not in an interactive field.

help index - A facility that allows the user to select topics for which help is available.

help panel - An information panel that is displayed in response to a "help" request from the user.

help window - A Common User Access defined secondary window that displays information when the user requests help.

Hertz - One cycle per second.

hex - Hexadecimal number

hit testing - The means of identifying which window is associated with which input device event.

hook - A mechanism by which procedures are called when certain events occur in the system. For example, the filtering of mouse and keyboard input before it is received by an application program.

hook chain - A sequence of hook procedures that are "chained" together so that each event is passed in turn to each procedure in the chain.

hot spot - (1) The area of a display screen that is activated to accept user input.

(2) The part of the pointer that must touch an object before it can be selected. This is usually the tip of the pointer.

Contrast with *action point*.

Glossary - I

icon - A pictorial representation of an item the user can select. Icons can represent items (such as a document file) that the user wants to work on and actions that the user wants to perform. In the Presentation Manager, icons are used for data objects, system actions and minimized programs.

icon editor - The tool provided in the Presentation Manager for creating icons.

IDC - Inter-Device-Driver Communication.

inactive window - In *Common User Access* architecture, a window with which the user is not currently interacting. The window cannot receive input from the keyboard. Contrast with *active window*.

in-memory buffer - A block of memory in the address space of the host machine, used for data transfer.

init time - See *initialization time, device driver*

initialization time, device driver - After the OS/2 loads a device driver, it sends it an OS/2 request packet to initialize. During this initialization, certain *DevHlp* functions are not permitted. Also called *init time*.

input focus - In *Common User Access* architecture, the area of a window where user interaction is possible from either the keyboard or the pointing device.

Input/Output Control (IOctl) - A system function that provides a method for an application to send device-specific control commands to a device driver.

Input/Output Permission Map (IOPM) - A bit map pointed to by the 386 TSS that controls on a per-port basis whether an input/output instruction causes a fault.

Input/Output Privilege Level (IOPL) - Allows part of a Ring 3 application or device driver to execute at Ring 0.

input router - OS/2 internal process that removes messages from the system queue.

interactive graphics - Graphic images that can be moved or manipulated by the user at a terminal.

interactive program. - A running program that can receive input from the keyboard or other other input device. Also called a *foreground program*. Contrast with

noninteractive program; compare with *active program*.

Inter-Device-Driver Communication (IDC) - A process where one device driver calls another device driver, usually to obtain information or to request a function to be performed.

interrupt - An instruction that directs the microprocessor to suspend what it is doing and run a specified routine. When the routine is complete, the microprocessor resumes its original work. See also *routine*.

interrupt request (IRQ) - Broadly, an "interrupt request level", referring to pending or in-service interrupt requests, or to a specific level (for example, [IRQ 4](#)).

interrupt request flag - A bit in the 8259 PIC controller that indicates an interrupt is pending on particular level. The VPIC also maintains a virtual interrupt request flag for each interrupt level for each DOS session.

interrupt service flag - A bit in the 8259 PIC controller that indicates an interrupt request is being serviced. It is cleared when the PIC is sent EOI. The VPIC maintains a virtual interrupt service flag indicating that a simulated interrupt is in-progress in a DOS session.

interrupt time - When a device driver is run because of an interrupt rather than because of an application request. OS/2 device drivers receive interrupts either from the hardware they manage or from the system real-time clock.

During interrupt time, certain *DevHlp* functions are not permitted. Also, addresses received directly from OS/2 applications might not be valid unless they are converted system addresses.

IOctl - Input/Output Control.

IOPL - Input/Output Privilege Level.

IOPM - Input/Output Permission Map.

IORB - Input/Output Request Block.

Input/Output Request Block (IORB) - A data structure defined by this specification that is passed as a parameter on all calls to an adapter device driver. It contains a fixed section, followed by a command-dependent section.

IORBH - Input/Output Request Block Header

IPC - InterProcess Communication

IRET - Interrupt REturn.

IRQ - Interrupt ReQuest.

ISA - Industry Standard Architecture

Glossary - J

J

journal - A special-purpose file that is used to record changes made in the system.

Glossary - K

K

kanji - A graphic character set used in Japanese and other oriental "ideographic" alphabets.

kernel - (1) The part of an operating system that performs basic functions such as allocating hardware resources.

(2) A program that can run under different operating system environments.

(3) The part of the AIX operating system for RISC system/6000 that are needed frequently.

kernel mode - In the AIX operating system, the state in which a process runs in kernel model. Contrast with *user mode*.

kerning - The design of graphics characters so that their character boxes overlap. Used to proportionally space text.

keys help - A facility that gives the user a listing of all the key assignments for the current application.

Glossary - L

L

language support procedure - Function provided by the Presentation Interface for applications that do not or cannot (as in the case of COBOL and FORTRAN programs) provide their own dialog or window procedures.

Last-In-First-Out (LIFO) - A data queuing scheme in which the next item to be retrieved is the item that has been in the queue for the shortest time.

latency - The time interval between the instant at which an instruction control unit initiates a call for data and the instant at which the actual transfer of the data starts. Synonymous with waiting time.

LCT - Logical Color Table.

LDT - Local Descriptor Table.

LIFO - Last-In-First-Out.

LIFO stack - A data structure from which data is retrieved in "Last-In, First-Out" order.

linked list - A list in which the data elements may be dispersed but in which each data element contains information for locating the next. Synonymous with *chained list*.

list box - A control window containing a vertical list of selectable descriptions.

list panel - A defined panel type that displays a list of items from which a user can select one or more choices and then specify one or more actions to work on those choices.

linear address - A 32-bit address for a byte of storage that might or might not have been assigned to physical storage. The host CPU automatically translates linear addresses to physical addresses and raises an exception if OS/2 must assign physical storage to a referenced linear address. Synonymous with *flat address*.

linked list - A list in which the data elements may be dispersed, but in which each data element contains information for locating the next. Synonymous with *chained list*.

Data Elements:

Info

Info

Info

Local Descriptor Table (LDT) - A table that defines code and data segments specific to a single task.

logical palette - An array of *RGB* and mapping index pairs, created by the device driver when defining a palette (as a result of a [GpqCreatePalette](#) call).

logical unit - A physical or virtual peripheral device addressable through a destination.

Logical Unit Number (LUN) - An encoded 3-bit identifier for a logical unit.

LUN - Logical Unit Number.

LVB - Logical Video Buffer.

Glossary - M

M

main window - The window that is positioned relative to the desktop window.

maximize - A window-sizing action that makes a window the largest possible size.

media window - The part of the physical device (display, printer or plotter) on which a picture is presented.

memory device context - A logical description of a data destination that is a memory bit map. See also *device context*.

menu - A type of panel that consists of one or more selection fields. Also called a "menu panel".

message - (1) In the Presentation Manager, a packet of data used for communication between the Presentation Interface and windowed applications.

(2) In a user interface, information not requested by the user, but presented by the computer in response to a user action or internal process.

Messages on status, problems or user actions from a computer application should be distinguished from a "message" or note sent to the user by other users over a communications link.

message filter - The means of selecting which messages from a specific window will be handled by the application.

message queue - A sequenced collection of messages to be read by the application.

metafile - A generic format for the definition of the contents of a graphic image. Metafiles can be used to allow graphic images to be used by other applications.

metafile device context - A logical description of a data destination that is a metafile which is used for graphics interchange. See also *device context*.

metalanguage - A language used to specify or describe another language.

mickey - A unit of measurement for physical mouse motion whose value depends on the mouse device driver that is currently loaded.

minimize - A window-sizing action that makes the window the smallest possible size. In the Presentation Manager, minimized windows are represented by icons.

mixed character string - A string containing a mixture of one-byte and *kanji* or *Hangeul* (two-byte) characters.

MMPM/2 - MultiMedia Presentation Manager/2.

mnemonic - A memory-reinforcing device, such as a highlighted letter on a pull-down menu item.

mode - A method of operation in which the actions available to the user are determined by the state of the system.

modal dialog box - The type of control that allows the operator to perform input operations on only the current dialog box or one of its child windows. Also called a *serial dialog box*. Contrast with *parallel dialog box*.

model - The conceptual and operational understanding that a person has about something.

modeless dialog box - The type of control that allows the operator to perform input operations on any of the application's windows. Also called a *parallel dialog box*. Contrast with *modal dialog box*.

mouse - A hand-held device that is moved around to position the pointer on the screen.

multiple DOS sessions - A system service that coordinates the concurrent operation of several DOS sessions. See also *DOS session*.

multiprogramming - A mode of operation that provides for interleaved, concurrent execution of two or more programs by a single CPU.

multiple-choice selection - A mode that allows the user to select any number of choices (including no choice at all). See also *check box*. Contrast with *extended-choice selection*.

multi-tasking - The concurrent processing of applications or parts of applications. A running application and its data are protected from other concurrently running applications.

mutex - mutually exclusive (semaphore)

Glossary - N

N

named pipe - A named object that provides client-to-server, server-to-client or duplex communication between unrelated processes. Contrast with *unnamed pipe*.

nibble - Part of a byte, usually one-half. Occasionally spelled "nybble".

NMI - Non-Maskable Interrupt

noninteractive program - A program that is running (active) but is not ready to receive input from the user. Compare with *active program*. Contrast with *interactive program*.

notification callout - The feature that provides for a routine to be called on completion of an input/output request. See also *notification routine*.

notification routine - The routine indicated in an input/output request block to be called on completion of that request. See also *notification callout*.

null-terminated string - A string of (*n*+1) characters where the (*n*+1)th character is the "null" character (X'00') and is used to represent an *n*-character string with implicit length. Also called a "zero-terminated" string or an "ASCIIZ". string.

Glossary - O

O

object - In *Common User Access* architecture, something that the user works with to perform a task. Text and graphics are examples of CUA objects.

object window - A window that does not have a parent but which may have *child windows*. An object window cannot be presented on a device.

OEM - Original Equipment Manufacturer.

open - To start working with a file, directory or other object.

optical memory - A storage device that uses an optical medium.

Original Equipment Manufacturer (OEM) - A manufacturer of equipment that may be marketed by another manufacturer.

OS/2 session - A session created by the operating system, that supports the independent processing of an OS/2 program.

output area - The area of the output device within which the picture is to be displayed, printed or plotted.

owner window - A window into which specific events that occur in another (owned) window are reported.

Glossary - P

P

paint - The action of drawing or redrawing the contents of a window.

palette - In *Common User Access* architecture, a list of colors assigned to various areas on a panel. The user can change these colors.

panel - A particular arrangement of information grouped together for presentation to the user in a window.

panel area - An area within a panel that contains related information. The three major panel areas defined by *Common User Access* architecture are the action bar, the function key area and the panel body.

panel body - The portion of a panel not occupied by the action bar, function key area, title or scroll bars. The panel body may contain protected information, selection fields and entry fields. The layout and content of the panel body determine the panel type.

panel body area - The part of a window not occupied by the action bar or function key area. The panel body area may contain information, selection fields and entry fields. Also called the *client area*.

panel body area separator - A line or color boundary that provides the user with a visual distinction between two adjacent areas of a panel.

panel definition - A description of the contents and characteristics of a panel. Thus, a panel definition is the application developer's mechanism for predefining the format to be presented in a window to the user.

panel ID - A panel element located in the upper left-hand corner of a panel body that identifies that particular panel within the application.

panel title - A panel element that identifies the information in the panel. Contrast with *titlebar*.

paper size - The size of paper, defined in either standard U.S. or European names (for example: *A*, *B*, *A4*) and measured in inches or millimeters respectively.

parallel dialog box - Synonymous with *modeless dialog box*.

parent process - A process that creates other processes. Contrast with *child process*.

parent window - A window in which another window is opened. Contrast with *child window*.

PDD - Physical Device Driver.

PDE - PageDirectoryEntry.

pel - *Picture Element*. In computer graphics, the smallest element of a display surface that can be independently assigned color and intensity (T), and switched between visible and invisible states. Synonymous with "display point", "picture element", and *pixel*.

phase alignment - Aligning source bits with destination bits. Often required in a *BitBlt* function move operation where byte blocks are moved on bit boundaries.

physical address - A 32-bit byte address giving the actual address in physical storage for a data item.

physical device driver (PDD) - A device driver responsible for primary control of a physical device. This corresponds very closely to the OS/2 1.00 dual-mode device driver, except that it does not support the DOS environment directly. Rather, it supports interfaces that can be used by virtual device drivers to support DOS applications.

pick - To select part of a displayed object using the pointer.

pipe - See *named pipe*, *unnamed pipe*.

picture element - Synonymous with *pel*.

PIO - Programmed Input/Output.

pixel - *PeI*

PM - Presentation Manager.

PMI - Protect-Mode Interface.

plotter - An output device that uses pens to draw its output on paper or transparency foils.

pointer - The symbol displayed on the screen that is moved by a pointing device such as a *mouse*. The pointer is used to point at items that the user can select. Contrast with *cursor*.

pointing device - A peripheral instrument such as a mouse, trackball, joystick, digitizer tablet, or paddle, used to control the position of a pointer on the screen.

pointings - Pairs of X-Y coordinates produced when a user defines positions on a screen with a *pointing device*.

polyfillet - A *fillet* arc, made up of a series of short lines. It is tangential to the end points of the first and last lines and also tangential to the

midpoints of all other lines in the sequence.

polyline - A line that appears to be a single line but which is actually composed of a series of connected short lines.

pop - To retrieve an item from a last-in-first-out stack of items. Contrast with *push*.

pop-up window - A window that appears on top of another window in a dialog. Each pop-up window must be completed before returning to the underlying window.

prefetch - To locate and load a quantity of data in anticipation of a request.

presence-check function - A Ring 3 (non-privileged) *.EXE* program that determines whether a given hardware interface is present on a workstation.

PRESENCECHECK - A keyword, interpreted by the DDINSTALL utility, to determine whether to process the device driver profile file, based on the return code from PRESENCECHECK.

Presentation Manager - The OS/2 control program plus the visual component that presents, in windows, a graphics-based interface to applications and files.

primary window - The window in which the main dialog between the user and the application takes place. In a multi-programming environment, each application starts in its own primary window. The primary window remains in place for the duration of the application, although the panel displayed will change as the user's dialog proceeds. See also *secondary window*.

print job - The result of sending a document or picture to be printed.

Print Manager - In the Presentation Manager, the part of the spooler that manages the spooling process. It also allows the user to view print queues and to manipulate print jobs.

privilege level - A method of protection supported by the hardware architecture of IBM personal computer systems in which only certain program instructions can be used by certain programs. This method consists of four "privilege levels" known as rings (numbered 0 through 3). Synonymous with *ring level*.

process - (1) An operation or combination of operations on data.

(2) The second level of increasing system granularity.

Compare with *session* and *thread*. See the illustration at *session*.

program details - Information about a program that is specified in the Desktop Manager window and is used when the program is started.

program group - In the Presentation Manager, several programs that can be acted upon as a single entity.

Programmed Input/Output (PIO) - The transfer of data between an adapter and the

host CPU using input and output instructions.

program name - The full file specification of a program. Contrast with *program title*.

program title - The name of a program as it is listed in the Desktop Manager window. Contrast with *program name*.

protect mode - In OS/2, a method of program operation that limits or prevents access to certain instructions or areas of storage. Contrast with real mode.

pull-down - In *Common User Access* architecture, pertaining to an extension of the *action bar* that displays a list of choices available for a selected action bar choice. After a user selects an action bar choice, the pull-down appears with the list of choices. Additional *pop-up windows* may appear from pull-down choices to further extend the available actions.

push - To add an item to a *LIFO* stack of items. Contrast with *pop*.

push button - In *Common User Access* architecture, a *control window* in the shape of a rounded-corner rectangle on the screen, containing associated text. Push buttons are used to initiate actions that occur immediately when the push button is selected, such as "enter" or "cancel".

Contrast with *radio button* and *check box*.

Glossary - Q

Q

queue - A list of print jobs waiting to be printed.

queued device context - A logical description of a data destination (for example, a printer or plotter) where the output is to go through the spooler. See also *device context*.

quiet failure - An initialization failure resulting when a device driver fails to recognize the hardware interface. Quiet failure prevents the generation of failure messages on the workstation display.

Glossary - R

R

radio button - In *Common User Access* architecture, a *control window* in the form of a small circle on the screen, with adjacent associated text. A radio button represents a single choice in a fixed set of choices, from which only one can be selected. The circle is partly filled when a choice is selected.

Contrast with *push button* and *check box*.

RBA - Relative Block Address.

Read-Only Memory (ROM) - A storage device in which data, under normal conditions, can only be read.

real mode - In VSE, a mode in which a program may not be paged.

reentrant - The attribute of a program or routine that allows the same copy of the program or routine to be used concurrently by two or more tasks.

reference phrase - A word or phrase that is emphasized in a device-dependent manner in order to inform the user that additional information for the word or phrase is available.

reference phrase help - "Help" information on a selectable phrase.

refresh - To update a window with changed information on its current status.

Relative Block Address (RBA) - The location of a block. Most direct access devices have an inherent unit of allocation (usually a *sector*) that is referred to as a *block* and considered the smallest addressable quantity on the device. Relative block addressing relates to a device composed of consecutively numbered blocks starting at 0 (zero).

RBA numbering is done without regard to any physical characteristics (cylinder/head/sector) organization the device might exhibit.

removable-media indicator - A flag (bit) indicating that a device permits media removal.

resource - The means of providing extra information used in the definition of a window. A resource can contain definitions of fonts, templates, accelerators and mnemonics; the definitions are held in a resource file.

restore - To return a window to its original size or position following a sizing

or moving action.

resurrection - The Presentation Manager event that occurs when switched back from a full-screen DOS or WIN-OS/2 session.

restore - To return a window to its original size or position following a sizing or moving action.

RETF - RETurn Far.

reverse video - A form of alphanumeric highlighting for a character, field or cursor in which its color is exchanged with that of its background. For example, changing a red character on a black background to a black character on a red background.

RGB - (1) Color coding in which the brightness of the additive primary colors of light, Red, Green, and Blue, are specified as three distinct values of white light.

(2) Pertaining to a Red/Green/Blue color display or display adapter..

ring level - See *privilege level*.

ROM - Read-Only Memory.

roman - Relating to a type style with upright characters.

ROP - Raster OPeration.

routine - A program, or part of a program, that may have some general or frequent use.

RPC - Remote Procedure Call.

RTC - Real-Time Clock.

Glossary - S

S

SCB - Subsystem Control Block architecture.

screen - The physical surface of a workstation or terminal upon which information is presented to the user.

screen device context - A logical description of a data destination that is a particular window on the screen. See also *device context*.

scroll bar - A control window, horizontally or vertically aligned, that allows the user to scroll additional data into an associated panel area.

scrollable entry field - An entry field larger than the visible field.

scrollable selection field - A selection field that contains more choices than are visible.

scrolling - Moving a display image vertically or horizontally in a manner such that new data appears at one edge as existing data disappears at the opposite edge.

SCSI - Small Computer System Interface.

seamless windows - An architecture contained within OS/2 which permits one or more applications to share windowed desktop graphical space and other resources, while executing concurrently. Application session windows managed by seamless windows can share border information, and pointing device transitions from session to session are handled smoothly and transparently.

secondary window - A type of window associated with the primary window in a dialog. A secondary window begins a secondary and parallel dialog that runs at the same time as the primary dialog. See also *primary window*.

second-party DMA adapter - See *DMA slave*.

sector - (1) A predetermined angular part of a track or band on a magnetic drum or magnetic disk that can be addressed.

(2) On disk or diskette storage, an addressable subdivision of a track used to record one block of a program or data.

segment - See *graphics segment*.

segment attributes - The attributes that apply to the segment as an entity, as opposed to the individual graphics primitive within the segment; for example, the visibility, transformability, or detectability of a segment.

Contrast with *graphics attributes*.

select - To mark or choose an item. Note that "select" means to mark or type in a choice on the screen, while "enter" means to send all selected choices to the computer for processing.

select button - The button on a pointing device, such as a mouse, that is pressed to select a menu choice. Also called "Button 1".

selection cursor - A type of cursor used to indicate the choice or entry field that the user wants to interact with. It is represented by highlighting the item it is currently positioned on.

selection field - A field containing a list of choices from which the user can select one or more.

semaphore - (1) A variable that is used to enforce mutual exclusion.

(2) An indicator used to control access to a file.

(3) An entity used by multi-threaded applications for signalling purposes and to control access to the *serially-reusable resources* of the system.

Processes can be locked to a resource with semaphores if the processes follow certain programming conventions.

sense data - Data which describes an I/O error as defined by the ANSI SCSI specifications.

sense data area - Storage for sense data.

separator - See *panel body area separator*.

serial dialog box - Synonymous with *modal dialog box*.

serially-reusable resource (SRR) - A logical resource or object that can be accessed by only one task at a time.

session - (1) One instance of a started process or logical group of processes. Each session is separate from all other sessions that might be running on the computer. The operating system is responsible for coordinating the resources that each session uses.

These resources can include computer memory, allocation of processor time, and windows on the screen.

(2) The third level of increasing system granularity:

thread

2

process

3

session

OS/2

World

Compare with *thread* and *process*.

session, DOS - See *DOS session*.

session, OS/2 - See *OS/2 session*.

session, WIN-OS/2 - See *WIN-OS/2 session*.

shadow box - An area on the screen that follows mouse movements and shows what shape the window will take if the mouse button is released.

shutdown - In the Desktop Manager, the procedure required before the computer is switched off to ensure that data is not lost.

sibling windows - *Child windows* that have the same *parent window*.

slider box - An area on the scroll bar that indicates the size, position, range, and properties of the visible information in a panel area in relation to the information available. Synonymous with *thumb mark*.

Small Computer System Interface (SCSI) - An input and output bus that provides a standard interface between the OS/2 multimedia system and peripheral devices.

SMV - Software Motion Video.

spline - See *Bezier curve*.

split device driver - A device driver that is separated into a hardware-independent portion and one or more device-dependent portions.

spooler - A program that intercepts the data going to printer devices, and writes it to disk. The data is printed or plotted when it is complete and the required device is available. The spooler prevents the intermixing of output from different sources.

SRR - Serially-Reusable Resource.

standard window - A collection of windows that form a panel.

static control - The means by which the application presents descriptive information (for example, headings and descriptors) to the user. The user cannot change this information.

suballocation - The allocation of a part of one extent for occupancy by elements of a component other than the one occupying the remainder of the extent.

Subsystem Control Block (SCB) architecture - An IBM-defined CPU-to-adapter protocol for communication with some of its adapters.

switch - An action that moves the input focus from one area to another. This can be within the same window or from one window to another.

switch list - See Task List.

symbolic identifier - A text string that equates to an integer value in an include file that is used to identify a programming object.

synchronous - Pertaining to two or more processes that depend upon the occurrence of specific events such as common timing signals.

System Menu - In the Presentation Manager, the pull-down in the top left corner of a window that allows it to be moved and sized with the keyboard.

system queue - The master queue for all pointer device or keyboard events.

Systems Application Architecture (SAA) - The *Common User Access* architecture, the *Common Programming Interface*, and the *Common Communication Support*.

Glossary - T

T

tag - A markup language word and its attributes that are entered in the source file to identify parts of a panel or other dialog objects.

Task List - In the Presentation Manager, the list of programs that are active. The list can be used to switch to a program and to stop programs.

template - An ASCII-text definition of an action bar and pull-down menu held in a resource file or as a data structure in program memory.

text - Characters or symbols. Contrast with *graphics*.

text cursor - A symbol displayed in an entry field that indicates where typed input will appear.

text window - Also known as the Video Input/Output (VIO) window. The environment in which OS/2 runs AVIO applications.

task time - Refers to an OS/2 device driver being called as the result of an application request, rather than due to a hardware interrupt.

thread - (1) The smallest unit of operation to be performed within a process.

(2) The first (or lowest, or finest) level of increasing system granularity.

Compare with *session* and *process*. See the illustration at *session*.

thumb mark - Synonymous with *slider box*.

thunk - Code used to "glue" 16- and 32-bit routines. A thunk can reside in an application or in a library module. A 32-bit thunk binds 32-bit code to 16-bit code. A 16-bit thunk binds 16-bit code to 32-bit code.

thunk layer - An interface that converts 32-bit parameters to 16-bit parameters, and maps linear addresses to segmented addresses.

tilde - A diacritical mark used to denote the character that is to be used as a *mnemonic* when selecting text items within a menu.

time slice - (1) The period of processing time allocated for running a program.

(2) An interval of time on the processing unit allocated for use in performing a task. After the interval has expired, processing time is allocated to

another task, so that a task cannot monopolize processing time beyond a fixed limit.

title bar - The area at the top of a window that contains the window title. The title bar is highlighted when that window has the input focus. Contrast with *panel title*.

tree - In the Presentation Manager, the window in the File Manager that shows the organization of drives and directories.

tuple - In a relational database, a part of a relation that uniquely describes an entity and its attributes.

Glossary - U

U

unchecked state - See *check box*.

uninstall - In products with OS/2 preinstalled, a choice offered to the user.

unnamed pipe - A circular buffer created in memory; used by related processes to communicate with one another. Contrast with *named pipe*.

User Shell - A component of OS/2 that uses a graphics-based, windowed interface to allow the user to manage applications and files installed and running under OS/2.

Glossary - V

V

VBIOS - Virtual BIOS device driver

VC MOS - Virtual CMOS device driver

VDD - Virtual Device Driver

VDH - Virtual video Device Handler

VDM - Virtual DOS Machine; use DOS session.

VDMA - Virtual Direct Memory Access device driver

VFLPY - Virtual (FLoPpY) diskette device driver

VDSK - Virtual hard DiSK device driver

VGA - Video Graphics Adapter

VIO - Video Input/Output.

VIRR - Virtual Interrupt Request Register

Virtual Device Driver (VDD) - (1) Virtualizes the hardware interfaces of an option adapter or device, usually to migrate an existing DOS application into an OS/2 DOS session.

(2) Essentially a Dynamic Link Library; a virtual device driver generally does not interface directly with the hardware.

(3) A 32-bit *.EXE* file that can contain initialization code, initialization data, and swappable global code.

(4) Maintains shadow state of hardware, if necessary. Allows a DOS session to execute in a *window* or in the *background* by intercepting direct device access and simulating that device.

virtual DevHlp (VDH) - Kernel (linear memory, paging, hardware interrupt, event control, port control) services provided to virtual device drivers.

virtual memory (VM) - Addressable space that is apparent to the user as the processor storage space but not having a fixed physical location.

Virtual Programmable Interrupt Controller (VPIC) - Virtualizes the 8259 Programmable Interrupt Controller (PIC). A special virtual device driver, in that it provides services to other virtual device drivers.

visible region - A window's presentation space clipped to the boundary of the window and the boundaries of any overlying window.

VKBD - Virtual KeyBoard device driver.

VLPT - Virtual parallel port device driver.

VNPX - Virtual numeric coprocessor device driver.

VPIC - Virtual Programmable Interrupt Controller device driver.

VRAM - Video Random-Access Memory.

VTIMER - Virtual TIMER device driver.

V86 mode - Virtual 8086 mode of the 80386 CPU.

Glossary - W

W

What You See Is What You Get (WYSIWYG) - A capability that enables text to be displayed on a screen in the same way that it will be formatted on a printed page.

wild-card character - The global file-name characters " ? " or " * ".

window - (1) A rectangular area of the display surface in which display images pertaining to a particular application can be presented. Different applications can be displayed simultaneously in different windows. A window can be smaller, the same size, or larger than the screen. Several windows can appear to overlap on the screen.

(2) An area of the screen with visible boundaries within which information is displayed.

(3) A division of the screen in which one of several programs being executed concurrently can display information.

window class - A grouping of windows whose processing needs conform to the services provided by one window procedure.

window coordinates - The means by which a window position or size is defined; measured in device units or *pels*.

window procedure - Code that is activated in response to a message.

window rectangle - The means by which the size and position of a window is described in relation to the desktop window.

window style - A set of properties that influence how events related to a particular window will be processed.

WIN-OS/2 session - A session created by the operating system, that supports the independent processing of a Microsoft Windows program.

workstation - A display screen together with attachments such as a keyboard, a local copy device or a tablet.

WYSIWYG - What You See Is What You Get.

Glossary - X

X

XGA - eXtended Graphics Adapter

Glossary - Y

Y

There are no glossary terms for this initial letter.

Glossary - Z

Z

Z-order - The order in which *sibling windows* are presented. The topmost sibling window obscures any portion of the siblings that it overlaps; the same effect occurs down through the order of lower sibling windows.

zoom - In graphics applications, to increase (zoom in) or decrease (zoom out) the size of picture.

IBM Trademark

Trademark of the IBM Corporation.

Trademarks

Trademark of the Intel Corporation.

Trademarks

Trademark of the Microsoft Corporation.